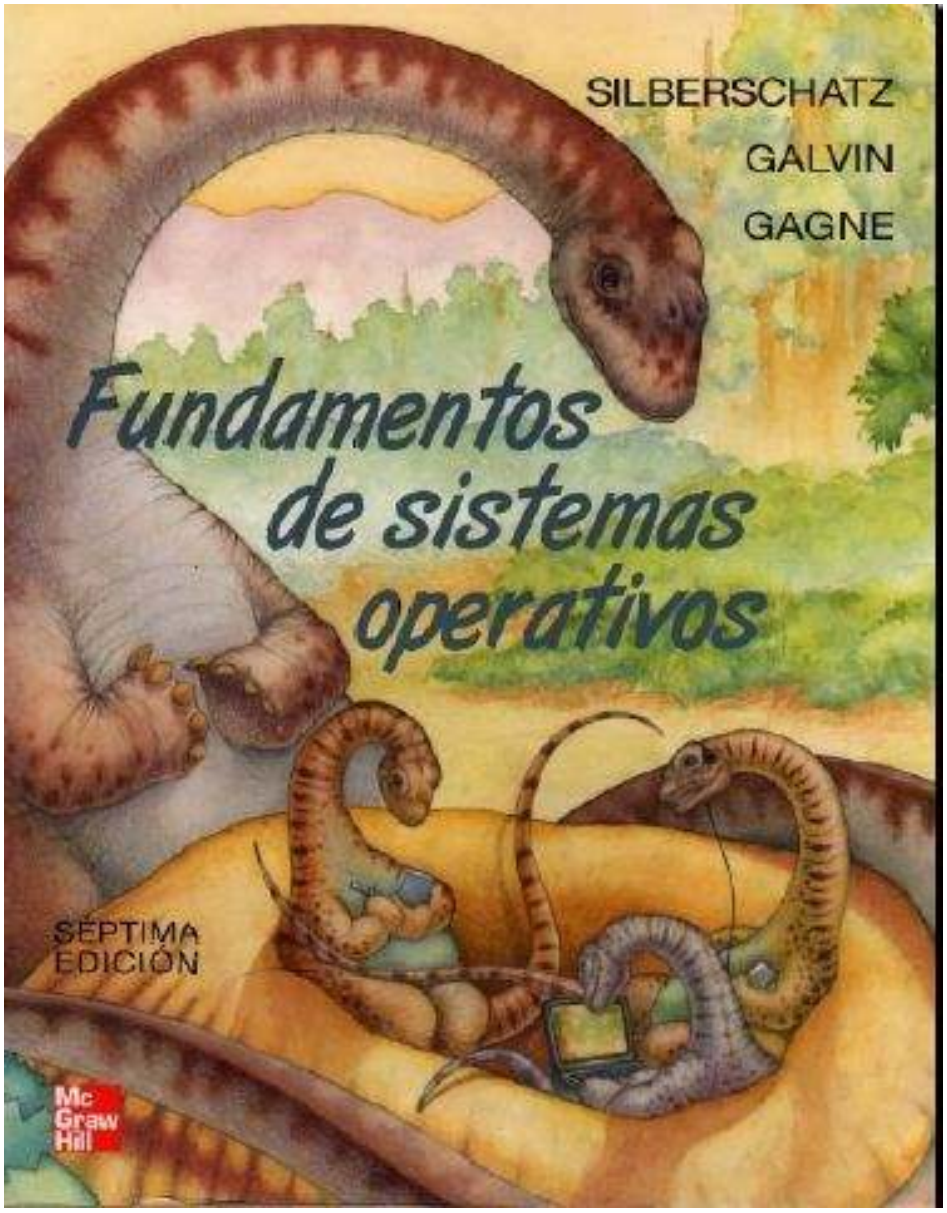
Resumen libro Silberschartz

Sistemas Operativos (Universidad Nacional de La Patagonia San Juan Bosco)

Resumen
Fundamentos de Sistemas Operativos
Silberschatz



Capitulo 2:
Estructura de Sistema operativo

Diseño e implementación de un sistema operativo:
Existen una serie de lineamientos a tener en cuenta para el diseño de sistemas operativos.

Objetivo del diseño:
En el nivel más alto el diseño del sistema se verá afectada por: la elección del hardware, y el tipo de sistema. Algunos de estos tipos de sistemas son:

- ✓ Procesamiento por lotes.
- ✓ Tiempo compartido.
- ✓ Monousuario.
- ✓ Multiusuario.
- ✓ Sistema Distribuido.
- ✓ Sistema de tiempo real.
- ✓ Sistema de propósito general.

Puede llegar a ser complicado especificar los requisitos, sin embargo hay dos grupos básicos:

- ✓ Objetivos del usuario.
- ✓ Objetivos del sistema.

Ambos presentan posturas vagas, en cuanto a cómo definir requisitos para el diseño, sin embargo no existe una única resolución para el diseño de un sistema operativo, aunque si existen una serie de principios generales en el campo de la Ingeniería del Software, que veremos a continuación.

Mecanismos y políticas:
Se debe separar políticas de mecanismos, unas determinan que hacer, y las otras determinan como hacerlo. Es importante por la cuestión de la flexibilidad, las políticas cambiaran mucho, en el peor de los casos modificaran mecanismos. Seria por lo tanto deseable un mecanismo general insensible a los cambios de política, uno de ellos entonces modificaría solamente algunos pequeños mecanismos. Los sistemas de microkernel son un ejemplo de esto, tienen un pequeño conjunto de bloques componentes primitivos, y estos son prácticamente independientes de las políticas y los mecanismos, por tanto estos se pueden adosar al kernel creador, según la dinámica del cambio, por el usuario, o a través de los propios programas de usuario.

Implementación:
Una vez diseñados estos deben implementarse, tradicionalmente esto se llevaba a cabo en lenguaje ensamblador, sin embargo en la actualidad se escriben en lenguajes de alto nivel como C o C++. El primer sistema que no fue escrito en lenguaje ensamblador fue probablemente MCP (Master Control Program), que fue desarrollado en el MIT, utilizando un lenguaje variante de Algol, llamado MULTICS. Los sistemas operativos como Linux y Windows, están escritos en su mayoría en C, aunque hay algunas secciones que están escritas en ensamblador, como las que controlan los dispositivos (controladores), y las que guardan y restauran el estado de los registros. Las ventajas de

usar un lenguaje de alto nivel o al menos un lenguaje de implementación de sistemas, consisten en que el código sea más compacto, más claro, pueda escribirse más rápido y se pueda depurar más fácil. Además el mejoramiento del compilador permite el mejoramiento del código generado para el sistema operativo mediante una simple recopilación, por ultimo un sistema operativo es más potable. Las únicas desventajas de implementar un sistema operativo en un lenguaje de alto nivel, son las cuestiones inherentes a la performance, sin embargo aunque los sistemas operativos tienen un gran tamaño, solo una pequeña parte del código resulta critica para conseguir mejorar dicho aspecto, estas secciones son básicamente el gestor de memoria principal, y el planificador de CPU. Luego de desarrollado y ejecutando, en el sistema se pueden identificar los cuellos de botella y reemplazarlos por las respectivas rutinas en ensamblador, para encontrarlos debemos monitorear el rendimiento del sistema.

Estructura del sistema operativo:
La ingeniería de un sistema es tan grande y compleja que debe hacerse cuidadosamente para que el sistema funcione apropiadamente y pueda modificarse con facilidad. Un método habitual consiste en dividir la tarea en componentes más pequeños en lugar de tener un sistema monolítico, cada uno de estos módulos debe ser una parte bien definida del sistema, con entradas, salidas, y funciones cuidadosamente especificadas. A continuación enumeraremos como alguno de los componentes vistos en el capítulo 1, se funden e interconectan en un kernel:

- ✓ Estructura simple:
  Muchos sistemas comerciales, no tienen una estructura bien definida, frecuentemente tales sistemas operativos comienzan siendo sistemas pequeños simples y limitados, y luego crecen más allá de su ámbito original, como MS-DOS.
- ✓ Estructura en niveles:
  Con el soporte de hardware apropiado, los sistemas operativos pueden dividirse en partes más pequeñas y más ordenadas que es lo que permitían los sistemas originales MS-DOS o UNIX. Con el método de diseño Top-Down se determinan las características y funcionalidades globales, y se separan en componentes, la ocultación de los detalles a los niveles superiores también es importante, un posible método de implementación es mediante niveles, el sistema operativo, se divide en una serie de capas (niveles), el nivel inferior o nivel “0” es el hardware, el nivel superior o nivel “n” es la interfaz de usuario. En un sistema de este tipo un nivel “x” podrá invocar rutinas de los niveles inferiores y podrá ser invocada por las rutinas de niveles superiores, cada nivel tendrá sus propios algoritmos y estructuras de datos. La principal ventaja de la elección de esta metodología es la fácil implementación y depuración. Por definición el único nivel que no afecta al resto del sistema en su depuración es el “0”, ya que solo utiliza el hardware básico (que se supone correcto) para implementar sus funciones, una vez depurada una de las capas se puede suponer correcta junto con las inferiores, lo que permite comenzar a depurar la siguiente, de encontrarse un error, se sabrá que este se solo se puede encontrar en el nivel en el que se está depurando, dado que los demás ya fueron depurados. Un nivel superior utiliza las rutinas y estructuras de datos provistas por el anterior y se abstrae de la implementación de ellas entendiendo únicamente su funcionalidad. El problema con esta metodología es la correcta división del sistema en capas, dado que cada una solo puede utilizar las operaciones y datos de la inferior, otro problema con

esta metodología es que los kernels diseñados con ella suelen ser menos eficientes en performance.

- ✓ Microkernel:
  La idea es eliminar todas las cosas que no sean esenciales del kernel e implementarlas como programas del sistema y del usuario, el resultado es un kernel más reducido, no existe consenso a cerca de cuando implementar una funcionalidad del sistema operativo como programa del sistema o aplicación, sin embargo los microkernels proporcionan una gestión de la memoria principal y de los procesos mínimamente, además de un mecanismo de comunicaciones. La función principal del microkernel radica en la comunicación entre el programa cliente y los distintos servicios que se ejecutan también en el espacio del usuario. Las ventajas de esta metodología son básicamente, la implementación del sistema operativo, su portabilidad debido a su pequeño tamaño, la seguridad, y la reducción del riesgo de caída del sistema a causa de la caída de un programa, producto del hecho de que no haya tantos procesos corriendo a nivel kernel.
- ✓ Módulos:
  Las técnicas de programación orientada a objetos, son tal vez la mejor metodología para modularizar un kernel. En este tipo de metodologías el kernel tendrá una serie de componentes fundamentales, que asignara dinámicamente. El resultado es similar al de niveles, en el sentido de que cada sección del kernel tiene interfaces bien definidas y protegidas, pero es más flexible porque cualquier modulo puede eliminar a cualquier otro modulo, y también tiene las ventajas del microkernel dado que el kernel termina siendo una sección muy pequeña de funciones esenciales, pero sin el hecho de transmitir mensajes entre las utilidades del sistema y las aplicaciones, y el microkernel como modo de comunicación.

Capitulo 3:
Procesos

Proceso: consiste básicamente en un programa en ejecución.

Existen dos tipos de procesos:

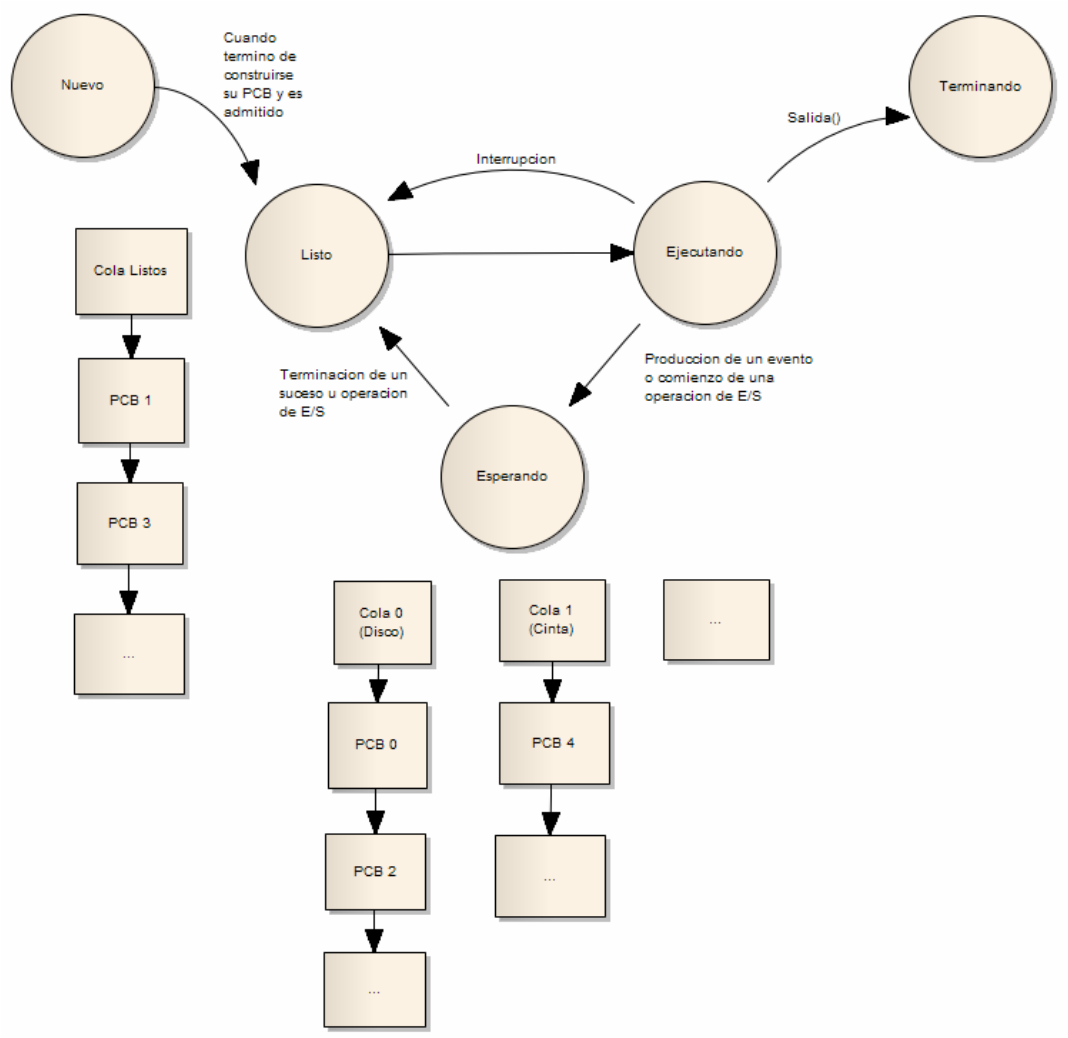
- ✓ Del sistema.
- ✓ Del usuario.

Un proceso contiene en memoria principal un conjunto de componentes, y en la CPU una serie de registros que definen su entorno. Los componentes en memoria son los siguientes:

- ✓ Heap: porción de memoria principal utilizada para el cúmulo de memoria dinámicamente asignada al proceso en tiempo de ejecución.
- ✓ Código: porción de memoria principal que se envía a ejecutar en el CPU.
- ✓ Datos: porción de memoria principal que contiene las variables globales del proceso.
- ✓ Pila: porción de memoria principal que contiene las variables locales, argumentos de las funciones, y direcciones de retorno.

En un proceso en memoria, la Pila crece de las direcciones más inferiores de la misma hacia las direcciones más superiores, y el Heap a la inversa.
Un programa es una entidad pasiva, nada más que una lista de instrucciones almacenadas en un dispositivo (por lo general el disco), por el contrario un proceso es una entidad activa, con una serie de registros que definen su entorno, y una serie de recursos asignados a su cargo.
Sin importar cuantas veces se ejecute un mismo programa, se considera a cada uno de esos procesos en memoria, un proceso individual, con todos sus componentes, por más que se repita código.

Ciclo de vida de un proceso:



Estados por los que transita:

- ✓ Nuevo: se está creando la PCB del programa y otras cosas.
- ✓ Ejecutándose: se están ejecutando las instrucciones.
- ✓ Listo: está en espera de asignación de CPU.
- ✓ Terminado: ha terminado la ejecución del proceso.
- ✓ Esperando: está en espera de que se produzca un evento (una E/S o una recepción de señal, etc.).

PCB (Process Control Block):
Es una estructura del sistema que posee múltiples elementos de información, asociados con un proceso específico, entre los cuales se encuentran:

- ✓ Estado de proceso: cualquiera de los antes mencionados.
- ✓ Contador de programa: el puntero de instrucción del proceso.
- ✓ Registros de CPU: todos los registros necesarios para que el proceso pueda seguir ejecutando luego de producida una interrupción.
- ✓ Información planificación CPU: aquella información pertinente a la planificación.

- ✓ Información de gestión de memoria: información acerca de los registros base y limite, las tablas de páginas o tablas de segmentos, dependiendo del mecanismo de gestión de memoria utilizado por el sistema operativo.
- ✓ Información contable: incluye la cantidad de CPU y de tiempo real empleados, los límites de tiempo asignados, los números de cuenta, el numero de trabajo o proceso, etc.
- ✓ Información del estado de E/S: la lista de dispositivos asignados al proceso, la lista de archivos abiertos, etc.

Planificación de procesos:
Antes de comenzar con el tema cabe definir algunos conceptos:

Multiprogramación: la idea de la multiprogramación, es tener varios procesos al mismo tiempo conmutando entre ellos para que la CPU siempre tenga algo que ejecutar.

Multitarea: la idea es aumentar la conmutación de procesos tan frecuentemente que le dé al usuario la sensación de pseudo-paralelismo de procesos, a pesar de que el sistema operativo disponga de un solo CPU para ejecutar.

En función de estos dos objetivos, el planificador ejecuta un proceso entre varios disponibles, logrando así la multiprogramación, y conmutando con mucha frecuencia se puede alcanzar la multitarea.

Colas de planificación:
Todos los distintos tipos se encuentran en memoria, y se clasifican en:

- ✓ Cola de procesos preparados (Cola de Ready): contiene todos los procesos listos para ser ejecutados, puede haber más una por prioridad.
- ✓ Cola de dispositivos (Cola de Wait): contiene todos los procesos en espera de un determinado dispositivo de E/S, es por esto que hay una de estas por dispositivo.

Una vez en ejecución el programa, se pueden producir los siguientes eventos:

- ✓ Realizar una operación de E/S, el proceso solicitante se coloca en una cola de dispositivo.
- ✓ Crear un proceso hijo y esperar a que termine.
- ✓ Desalojo del proceso de la CPU por una interrupción, poniéndolo de nuevo en la cola de procesos preparados.

Cuando termina el proceso se elimina de las colas y se desasignan su PCB y recursos. Cabe aclarar que cuando hay mas procesos de los que pueden ser ejecutados de forma inmediata preparados, estos se almacenan en una cola aparte, en un dispositivo de almacenamiento masivo (generalmente el disco). Poseen un planificador a parte llamado “Planificador a largo plazo”, que es el encargado de cargar alguno de estos procesos en la cola de preparados en memoria, la cual es planificada por el llamado “Planificador a corto plazo”, este ultimo debe ser muy rápido, dado que en el tiempo de selección del proceso a ejecutar, no se está ejecutando nada más que este.

Cambio de contexto:
Se produce al interrumpir el CPU, el sistema operativo guarda el contexto actual del proceso en su PCB y ejecuta la rutina que satisface la interrupción, el almacenado en la PCB, se realiza para garantizar que se volverá a ejecutar el programa que fue interrumpido. Para realizar un cambio de contexto se salvaguarda el contexto del proceso actual en el PCB como habíamos indicado, y luego se restaura otro PCB para ser ejecutado.

Operaciones sobre procesos:
Las operaciones que se pueden realizar con los procesos son las siguientes:

Creación de procesos: un proceso puede ser "padre" de otros procesos, y aquellos que genere se llamaran "hijos". La mayoría de los sistemas operativos identifican los procesos mediante dos identificadores que son normalmente números enteros, uno es el PID (Process Identifier) propio del proceso, y el otro PPID (Parent Process Identifier), propio del padre. Un proceso padre tendrá ciertos recursos asignados al momento de generar un proceso hijo, este podrá obtener sus propios recursos del sistema operativo o podrá estar restringido a un subconjunto de los recursos del padre, este puede tener que repartir recursos entre sus hijos o compartir algunos con ellos. También un proceso padre puede enviarle al proceso hijo al momento de crearlo, datos de inicialización (entrada).
Hay dos tipos de ejecución cuando un proceso genera otro:

- ✓ El padre continúa ejecutándose concurrentemente con el hijo.
- ✓ El padre espera hasta que alguno de sus hijos o todos hayan terminado de ejecutarse.

También existe dos posibilidades en cuanto al espacio de direcciones del proceso hijo:

- ✓ El hijo es un duplicado del padre.
- ✓ El hijo carga un nuevo programa.

Terminación del proceso:
Un proceso termina cuando ejecuto su ultima instrucción y realiza la llamada al sistema Exit(). Aquí es donde si el proceso devuelve un valor al padre del mismo, este puede enterarse mediante la llamada al sistema Wait() (esperar a que uno, varios o todos los hijos finalicen), en ese momento el sistema operativo dispone todos sus recursos.
También la terminación de un proceso puede ser el producto de que otro proceso haya pedido que este finalice, normalmente el único que puede realizar esta clase de operación es el padre del proceso. Casos por los cuales un padre podría terminar con un hijo:

- ✓ El hijo ha excedido el uso de algunos de los recursos.
- ✓ La tarea asignada al hijo ya no es necesaria.
- ✓ El padre abandona el sistema y el sistema operativo no permite que un proceso hijo siga si su padre ha terminado.

Es decir aclarando este ultimo ítem, si un padre termina, sus hijos también (terminación en cascada).

Comunicación entre procesos:
Los procesos se pueden clasificar en:

- ✓ Independientes: son aquellos procesos que no se comunican, es decir, no pueden ser afectados o afectar a los restantes procesos que se ejecutan en el sistema
- ✓ Cooperativos: son aquellos que si se comunican.

Razones de la cooperación entre procesos:

- ✓ Compartir información: entre usuarios.
- ✓ Acelerar cálculos: más de un proceso o hebra realizando una misma tarea, hace que esta por el procesamiento paralelo se realice más rápidamente.
- ✓ Modularidad: puede que un sistema sea un conjunto de procesos o hebras.
- ✓ Conveniencia: por el simple hecho de querer realizar varias tareas con un mismo recurso.

La cooperación entre procesos requiere necesariamente mecanismos de IPC (Inter Process Comunication), de los cuales hay dos modelos:
Memoria Compartida: requiere que los procesos establezcan una región de memoria, dispuesta para compartir información, normalmente reside en el espacio de direcciones del que la creo. En esta región el sistema operativo no interviene para solucionar conflictos, los procesos que se van a comunicar deciden donde estarán los datos y su formato.
Paso de mensajes: El sistema operativo proporciona los medios para que los procesos se comuniquen mediante el paso de mensajes, para ello en general el sistema operativo provee a los procesos de dos operaciones Send(), y Receive(). Para que los procesos puedan comunicarse además deben establecer un enlace de comunicaciones. Hay varias formas de implementar las operaciones de envío y recepción:

- ✓ Comunicación directa o indirecta.
- ✓ Comunicación sincronía o asíncrona.
- ✓ Almacenamiento en búfer explicito o automático.

Nombrado:
Si es una comunicación directa el emisor y receptor deben explicitar a su interlocutor:

Send(P,Mensaje);

Receive(Q,Mensaje);

Características:

- ✓ Los procesos conocen sus identidades.
- ✓ Un enlace, solo dos procesos.
- ✓ Dos procesos, un solo enlace.

El esquema anterior es simétrico, uno que fuese asimétrico, plantearía las primitivas de la siguiente forma:

Send(P,Mensaje);

Receive(&Id,Mensaje); (a “Id” se le asigna el nombre de cualquier proceso que envíe algo).

Si es comunicación indirecta, los mensajes se envían y reciben en un buzón o puerto (lógico), cada buzón tiene una identificación univoca. Ambos procesos deben establecer un buzón compartido:

Send(A,Mensaje);

Receive(A,Mensaje);

Características:

- ✓ Los procesos envían y reciben en un buzón.
- ✓ Los tipos de enlaces que se pueden establecer según como se recibe son:
  - Máximo dos proceso por enlace.
  - Solo un proceso recibe a la vez.
  - El sistema selecciona arbitrariamente quien recibirá.

El buzón puede ser propiedad del sistema operativo o del proceso. Si es del proceso no habrá problemas en quien recibe, solamente podrá recibir el propietario, y si es del sistema operativo (por lo tanto independiente de cualquier proceso), el sistema operativo debe proveer de los siguientes mecanismos al proceso:

- ✓ Crear buzón.
- ✓ Enviar y recibir mensajes a través del buzón.
- ✓ Eliminar buzón.

En principio en este ultimo esquema el propietario, será el único que podrá recibir y enviar, pero mediante la correcta llamada al sistema, se podrá conseguir múltiples receptores en cada buzón.

Sincronización:
Como antes mencionamos las opciones de envío y recepción, podrían ser síncronas o asíncronas:

- ✓ Envío con bloqueo: el proceso bloquea hasta que el proceso receptor o buzón recibe el mensaje.
- ✓ Envío sin bloqueo: envía el mensaje y continúa operando.
- ✓ Recepción con bloqueo: el receptor se bloquea hasta que hay un mensaje disponible.
- ✓ Recepción sin bloqueo: el receptor extrae un mensaje valido o nulo.

Almacenamiento en búfer:
Los mensajes que se envían o reciben siempre residen en una cola temporal, hay tres formas de implementarlas:

- ✓ Capacidad 0: la cola tiene una longitud máxima de “0”, entonces el emisor debe bloquearse hasta que el receptor reciba el mensaje.

- ✓ Capacidad limitada: la longitud es “n”, el emisor solo se bloquea si la cantidad de mensajes en la cola es “n”.
- ✓ Capacidad ilimitada: el emisor nunca se bloquea porque la cola es potencialmente infinita.

Estas últimas dos alternativas son consideradas de búfer automático, y la primera, sin búfer.

Capitulo 4:
Hebras

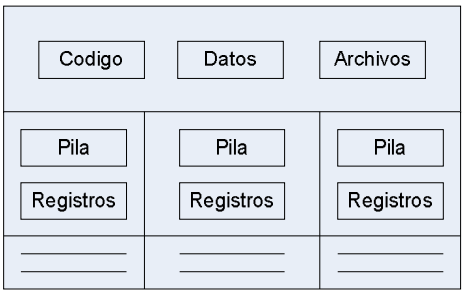
Hebra: unidad básica de utilización de la CPU.

Componentes de una hebra:

- ✓ Contador de programa.
- ✓ Un conjunto de registros.
- ✓ Una pila.
- ✓ Identificador de hebra.

Proceso tradicional:

Proceso multihebraico:



Ahora muchos kernels de sistemas operativos, utilizan la multihebra como recurso.

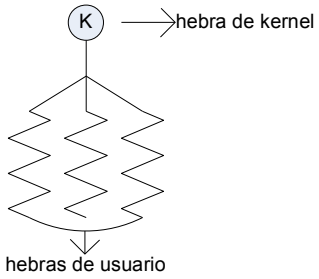
Ventajas de la programación multihebraica:

- ✓ Capacidad de respuesta: una hebra individual puede seguir ejecutando, a pesar de que parte del proceso este bloqueado o realizando una operación muy larga, lo que incrementa la capacidad de respuesta al usuario.
- ✓ Compartición de recursos: se comparten recursos como código, datos y archivos en un mismo proceso entre varias hebras, lo que permite en un mismo espacio de direcciones realizar no una sino múltiples actividades, sin necesidad de utilizar más memoria.
- ✓ Economía: la creación de procesos es costoso en cuanto a memoria, y recursos, dado que las hebras comparten los recursos del proceso es más fácil crear y realizar cambios de contexto entre unas y otras hebras.
- ✓ Utilización sobre arquitectura multiprocesador: las hebras tienen la posibilidad de ejecutarse en paralelo en este tipo de arquitectura.

Tipos de hebras:
Existen dos tipos ULT's (User Level Threads), KLT's (Kernel Level Threads), los del primer tipo no son reconocidos por el sistema operativo, y el soporte para ellas se proporciona por encima del kernel, mientras que en el otro caso el sistema operativo soporta y gestiona directamente las hebras.

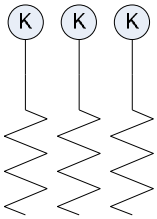
Relaciones entre las KLT y las ULT:

- ✓ Modelo muchos a uno:



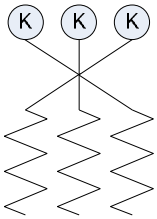
La gestión de hebras se hace mediante la biblioteca de hebras en el espacio del usuario, lo que lo hace más eficiente, pero el proceso entero se bloquea si una hebra lo hace, además como el sistema operativo no las reconoce, tampoco podrán ejecutar en paralelo en una arquitectura multiprocesador.

- ✓ Modelo uno a uno:



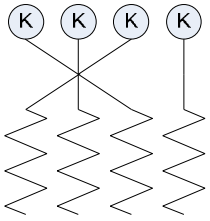
Proporciona una mayor concurrencia, permitiendo que se ejecuten múltiples hebras en paralelo en sistemas multiprocesador, el único inconveniente, es que para crear una hebra de usuario se hace necesario crear una kernel, y puede disminuir el rendimiento.

- ✓ Modelo de muchos a muchos:



Muchas hebras de usuario sobre un número menor o igual de hebras de kernel. La cantidad de hebras de kernel varía con respecto a la aplicación o a la arquitectura en la que se ejecute. Posee las ventajas del modelo uno a uno dado que si una hebra bloquea o desea ejecutar en otro procesador puede hacerlo utilizando una de kernel, pero permite al desarrollador crear múltiples hebras de usuario, obteniendo así una mejora en la performance.

- ✓ Modelo de dos niveles:



Combina los modelos muchos a muchos, y una hebra del modelo uno a uno.

Bibliotecas de hebras:

- ✓ De usuario: trabajan sobre el espacio de usuario, le proveen una API al desarrollador, que no posee funciones de creación y gestión de hebras mediante llamadas al sistema operativo, por ende las estructuras de datos y el código, se encuentran en el espacio de memoria del usuario.
- ✓ De kernel: trabaja sobre el espacio de kernel, le provee una API al desarrollador que posee funciones de creación y gestión de hebra, que se encuentran en el espacio de memoria de kernel.

Consideraciones de las hebras:

- ✓ Las llamadas de Fork(), y Exec():
  La llamada a Fork(), según el sistema operativo, duplicara o no el número total de hebras del proceso que invoco la función, o en algunos casos Fork(), tendrá un modo de duplicación total y otro para no duplicar (es decir solo la hebra principal), la cual sería la que realizo la llamada a Fork()). En el caso de Exec(), no habrá diferencias si el proceso es multihebraico o no lo es, en ambos casos reemplazara todo el proceso por uno nuevo sin respetar las hebras que tenia.
- ✓ Cancelación:
  La cancelación de una hebra objetivo, puede darse de dos formas diferentes:
  - Cancelación asíncrona: una hebra cancela a otra (la hebra objetivo) de forma inmediata.
  - Cancelación diferida: una hebra objetivo controla periódicamente si debe culminar, lo que le permite terminar de forma ordenada, decimos esto porque puede que la hebra objetivo deseara realizar algunas tareas antes de terminar, como liberar recursos o almacenar datos pertinentes para el resto de las hebras, etc.
- ✓ Tratamiento de señales:
  Se utilizan para notificar a un proceso que se produjo un determinado suceso. Existen dos tipos de señales síncronas, y asíncronas, y ambas siguen el mismo patrón:
  - Se genera una señal debido a un evento.
  - La señal se le suministra al proceso.
  - El proceso realiza lo necesario para tratarlo.

  Lo que diferencia las síncronas de las asíncronas, es que las primeras las produce el mismo proceso, por ende son síncronas con su flujo de control, por ejemplo la división por cero, el evento lo produce el mismo proceso, por el contrario las asíncronas son producto de un evento externo al proceso, por ejemplo pulsar la combinación de la tecla “control” mas la letra “c” del teclado. Cada señal puede ser tratada por una de dos posibles rutinas:
  - Una rutina predeterminada según la señal.
  - Una rutina definida por el usuario.
  - Esto puede complicarse en el segundo caso cuando se trata de un proceso multihebraico, dado que podría no saber a quién entregarle la señal, a que hebra claro está. Las opciones existentes para suministrarle a una hebra determinada dentro de un proceso una señal enviada a ese proceso son:
  - Suministrarle la señal a una hebra determinada.
  - Suministrarle la señal a todas las hebras.

- Suministrarle la señal a determinadas hebras.
- Suministrarle la señal a una hebra especifica que siempre reciba las señales.

✓ Conjunto de hebras:
Se utiliza para mejorar la performance, en los casos extremos donde el tiempo que se toma el sistema operativo para crear una hebra es necesario o donde hay tantas hebras corriendo simultáneamente que agotan los recursos del sistema, entre otras cosas. La idea básica es crear un conjunto de hebras al comienzo del proceso, que serán reutilizables, es decir se les asigna una tarea, la realizaran y luego, en vez de finalizar en ese momento su ciclo, volverán al conjunto de hebras compartidas, de no haber hebras disponibles, el proceso espera hasta que las haya.

✓ Datos específicos de una hebra:
Llamamos así a aquellos datos de los cuales una hebra especifica dentro de un proceso, solicita tener una copia propia única de ella con respecto a las demás hebras.

✓ Activación del planificador:
Existen los llamados LWP (o Light Weight Process) que sirven como comunicadores entre las hebras de usuario y las de kernel. Planifica este un conjunto limitado de hebras de usuario y ejecuta solo una de ellas, dado que cada LWP está asociado a una hebra de kernel. Puede ser necesaria una cierta cantidad de estos LWP para que un proceso trabaje de forma eficiente.

Capitulo 5:
Planificación de la CPU

Conceptos Básicos:
Como se dijo en capítulos anteriores, la idea de la multiprogramación es tener continuamente varios procesos en ejecución, para lógralo, se mantiene más de un proceso en memoria a la vez, y cuando uno de ellos debe esperar, el sistema operativos le otorga la CPU a otro.

Ciclo de ráfagas de CPU y E/S:
Antes de comenzar esta sección enunciaremos dos conceptos necesarios para continuar:

Ráfaga de CPU: es una secuencia de instrucciones que no implican realizar una operación de E/S en alguna parte de la secuencia.
Ráfaga de E/S: es una secuencia de instrucciones que implica realizar una operación de E/S en alguna parte de la secuencia.

La planificación de proceso depende de una propiedad observada de los mismos. Un proceso conmuta entre "Ráfagas de CPU" y "Ráfagas de E/S", hasta que finalmente en una de estas "Ráfagas de CPU", el proceso le solicita al sistema operativo terminar la ejecución. Las estadísticas realizadas al respecto, permiten seleccionar con un criterio un algoritmo de planificación correcto.

Planificación a corto plazo:
Existe un modulo del sistema llamado "Planificador de CPU" o "Planificador a corto plazo", encargado de seleccionar un proceso de la cola de preparados (Cola de Ready), y asignarle la CPU para que comience su ejecución, cabe aclarar que todos os proceso de esta cola, se encuentran ya cargados en memoria principal, y que lo que esta encolado en todas los distintos tipos de colas que se verán en el capitulo, serán PCB's.

Planificación Apropiativa:
A continuación, mostraremos cuatro circunstancias distintas en el ciclo de vida de un proceso en las que podrían ser necesarias tomar decisiones de planificación:

- ✓ Cuando un proceso cambia del estado de ejecución al de espera (ejemplo: realiza una operación de E/S, o espera por un hijo, etc.).
- ✓ Cuando un proceso cambia del estado de ejecución al de listo (Ejemplo: cuando se produce una interrupción, etc.).
- ✓ Cuando un proceso cambia del estado de espera al de listo (Ejemplo: al completarse una operación de E/S, etc.).
- ✓ Cuando un proceso termina.

En la primera y última situación el sistema operativo no tiene otra opción que ejecutar un nuevo proceso de la cola de preparados, sin embargo en las dos entre medio de las anteriores, se puede decidir planificar o no, si no se planifican, hablamos de un algoritmo de planificación "No Expropiativo", y si se planifican hablamos de un algoritmo "Expropiativo".

Despachador:
Es el modulo del sistema operativo encargado de otorgarle el control de la CPU a los procesos seleccionados por el planificador a corto plazo, las implicancias de su función son:

- ✓ Realizar el cambio de contexto.
- ✓ Realizar el cambio a modo usuario.
- ✓ Saltar a una posición correcta dentro de un programa de usuario para reiniciarlo.

El tiempo que tarda este modulo en detener un proceso e iniciar otro (que deberá ser lo más corto posible), se denomina "Latencia de Despacho".

Criterios de planificación:
Se han definido una serie de criterios para analizar cuál es el mejor de los algoritmos de planificación de CPU:

- ✓ Utilización de la CPU: implica el porcentaje de la CPU utilizado.
- ✓ Tasa de procesamiento: implica el cociente entre la cantidad de procesos completados y las unidades de tiempo que le llevo al sistema hacerlo.
- ✓ Tiempo de ejecución: implica para un proceso individual, cuantas unidades de tiempo tarda en ejecutarse, tomando como punto inicial la espera que realiza al cargarse en memoria principal, y como punto final su terminación sin incluir esta última.
- ✓ Tiempo de espera: implica la sumatoria de los periodos invertidos por un proceso en esperar en la cola de procesos preparados.
- ✓ Tiempo de respuesta: es el tiempo que tarda un proceso interactivo con el usuario, en realizar su primera respuesta al mismo, es decir en que tenga la respuesta disponible.

El objetivo del planificador es siempre maximizar las primeras dos, y minimizar las otras.

Algoritmos de planificación:
Consisten en decidir cuál será el próximo proceso de la cola de procesos preparados (Cola de Ready), al que se le asignara la CPU, los existentes son:

- ✓ FCFS (First Come First Served, o FIFO): se trata de un algoritmo no expropiativo, en el cual el primer proceso que se encola en la cola de preparados, será también el primero en ser atendido. La ventaja principal consiste en la sencillez de su implementación. La desventaja principal es en el tiempo de espera medio, el cual no será generalmente mínimo, y podrá variar significativamente si la duración de las ráfagas de CPU de los procesos son muy variables, sin contar que al ser no expropiativo, en sistemas de tiempo compartido donde es importante que la CPU conmute con frecuencia entre los diferentes usuarios, podría llegar a pasar que el CPU fuese retenido únicamente por un proceso durante un periodo largo de tiempo.

- ✓ SJF (Shortest Job First): se trata de un algoritmo que consiste en seleccionar de la cola el proceso que menor ráfaga de CPU tenga por ejecutar, para realizar esta selección este algoritmo le asocia a cada proceso en la cola de preparados la

duración de su siguiente ráfaga de CPU a ejecutar, el empate en la duración por ráfaga de CPU, se rompe mediante FCFS. Como principal ventaja tiene que el tiempo medio de espera es más corto para un conjunto dado de procesos, pero posee la dificultad de cómo determinar la duración de la siguiente ráfaga de CPU de cada proceso. Es por esta dificultad que por lo general se lo utiliza para planificar a largo plazo, y no a corto, dado que se podría tomar como estimador el límite de tiempo del proceso que el usuario especifique en el momento de enviar el trabajo. En el caso del corto plazo no hay forma de conocer la duración de la siguiente ráfaga de cada proceso, la única alternativa consiste en estimar dicho plazo, y esto se puede realizar con la siguiente ecuación:

$$\zeta_{n+1} = \alpha\, t_n + (1 - \alpha)\, \zeta_n$$

Donde:

ζ_{n+1} : es la duración predicha de la siguiente ráfaga de CPU del proceso a ejecutar.

t_n : es la duración real de la anterior ráfaga de CPU del proceso ejecutado.

ζ_n : es la duración predicha de la anterior ráfaga de CPU del proceso ejecutado anteriormente.

$0 < \alpha < 1$: es un parámetro que controla la importancia de la predicción anterior con relación al historial reciente, dado que si adquiere valores cercanos "0" entonces la duración predicha de la siguiente ráfaga de CPU tiende a ser igual que la predicha para la anterior ráfaga de CPU, y si adquiere valores cercanos a "1" entonces la duración predicha de la siguiente ráfaga de CPU tiende a ser igual que la duración real de la anterior ráfaga de CPU.

Este algoritmo puede ser tanto expropiativo, como no expropiativo, dependiendo como se encara la situación en la cual llega a la cola de procesos preparados, un proceso con ráfaga de CPU menor que el proceso que se encuentra ejecutando en dicho momento, si es expropiativo detendrá el proceso actualmente en ejecución y mandara a ejecutar el que acaba de llegar, mientras que si no lo es permitirá que dicho proceso finalice con su ráfaga de CPU.

- ✓ Planificación por prioridades: el algoritmo anterior es un caso particular de este, que consiste en asociar a cada proceso una prioridad, y ejecutar aquel que tenga la mayor, dado que SJF es un caso de prioridad, donde la misma predicción de la duración de la siguiente ráfaga de CPU es el inverso de la prioridad, cuando más grande la duración de la ráfaga de CPU menor la prioridad, y viceversa. No existe convención sobre si la prioridad más alta se corresponde con el número más alto o con el más bajo, sin embargo asumiremos que prioridades altas se corresponden con números bajos. Como en el caso de SJF el algoritmo de planificación por prioridades puede ser tanto expropiativo como no, y el caso para decidir si se emplea la expropiatividad o no es también el mismo que para el algoritmo anterior, se compara el proceso que acaba de arribar a la cola de preparados, con el que se estaba ejecutando, y si tiene menor prioridad que el que acaba de arribar, si es expropiativo le expropiara la CPU, y si no lo es no.

Una de las mayores desventajas del algoritmo de prioridades es la inanición que produce, una posible solución consiste en aplicar mecanismos de envejecimiento, donde se va aumentando gradualmente la prioridad de los procesos.

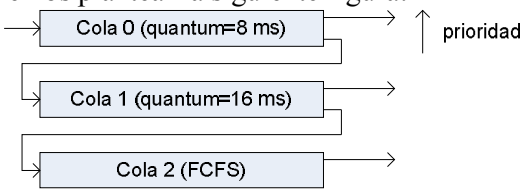
- ✓ Planificación por turnos: este algoritmo también se conoce como RR (Round Robin), y está diseñado para los sistemas exprópiatelos, básicamente es un algoritmo FCFS, pero con desalojo por tiempo, la duración de este es la de un cuanto de tiempo, en general esta magnitud es del orden de los 10 a 100 milisegundos. A cada proceso de la cola de preparados se le asigna un cuanto de tiempo para que ejecute. Una desventaja es el tiempo de espera medio, que por lo general será largo. El rendimiento de este algoritmo depende enormemente del tiempo que se le asigne al cuanto de tiempo, si es muy grande RR se convierte en FCFS, si es muy pequeño genera la apariencia de que cada uno de los procesos tiene su propio procesador, pero se deberá tener en cuenta el efecto de cambio de contexto en el rendimiento del sistema, es por esto que siempre el valor del cuanto de tiempo deberá ser mayor que el tiempo de cambio de contexto. Es también conveniente mencionar que si se aumenta el cuanto de tiempo no necesariamente mejora el tiempo medio de ejecución, y que tampoco se debe aumentar tanto dado que como se dijo antes podría el algoritmo transformarse en un FCFS.

- ✓ Planificación mediantes colas multinivel: este algoritmo divide la cola de preparados en varias colas distintas, y los procesos se van colocando en cada una de ellas según alguna propiedad del proceso (tamaño de memoria, prioridad del proceso, tipo de proceso, etc.), cada cola tiene su propio algoritmo de planificación, y hay una planificación general para todas las colas de preparados, esta ultima será por lo general expropiativa y de prioridad fija. A modo de ejemplificación de este algoritmo, se tiene la siguiente figura:

En el ejemplo, ningún otro proceso que no sea de la cola del sistema podrá ejecutarse hasta que dicha cola este vacía, y a medida que la cola tiene menos prioridad debe esperar a que se vacíen mayor cantidad de colas de mayor prioridad que ella. Además, si llegase un proceso de una cola de mayor prioridad que la cola del que se está ejecutando, dicho proceso será desalojado. También podrá emplearse una planificación por tiempo de CPU, otorgándoles a las colas de mayor prioridad mayor tiempo de CPU, para que cada cola administre su tiempo de CPU asignado, según su planificación particular.

- ✓ Planificación mediante colas multinivel realimentadas: el algoritmo anterior posee la ventaja de la disminución de la carga de trabajo de planificación, pero es poco flexible. La alternativa a esta rigidez del algoritmo anterior consiste en

separar los procesos en función de las características de sus ráfagas de CPU, permitiendo que un proceso se ubique en una cola tanto como en otras. Mientras mayor el tiempo de CPU, menor la prioridad y si se produce inanición de algún tipo, se puede pasar un proceso de una cola de baja prioridad a una más alta, este es también un mecanismo de envejecimiento. A modo de ejemplificación de este algoritmo podremos plantear la siguiente figura:



En el ejemplo si un proceso tiene una ráfaga de CPU mayor que 24 milisegundos, y empieza por ejecutar desde la cola "0" los primeros 8 milisegundos que le corresponden, luego será insertado al final de la cola "1", y cuando ejecute toda la cola "0" y sea su turno en la cola "1", ejecutara otros 16 milisegundos mas, y será colocado al final de la cola "2", donde ejecutara el tiempo que le reste, sea cuanto sea luego de que ejecuten las colas "0" y "1", y sea su turno en la "2"; un algoritmo de este tipo se define mediante los parámetros siguientes:

- Numero de colas.
- Algoritmo de planificación de cada cola.
- Método usado para determinar cuándo se pasa un proceso a una cola de prioridad más alta.
- Método usado para determinar cuándo se pasa un proceso a una cola de prioridad más baja.
- Método utilizado para determinar en qué cola se introducirá un proceso cuando haya que darle servicio.

La ventaja principal es que es el algoritmo mas general (se lo puede adaptar a cualquier sistema), pero también es el más complejo de implementar.

Planificación de sistema multiprocesador:
En un sistema de múltiples CPU se hace más compleja la planificación, y para abordarlo asumiremos que cada uno de dichos CPU's son homogéneos, en cuanto a su funcionalidad.

Métodos de planificación en sistemas multiprocesador:
El primer método consiste en que toda la planificación, el procesamiento de E/S y otras actividades del sistema sean gestionadas por un único procesador, "El Servidor Maestro". Los demás CPU's ejecutaran código de usuario, se denomina a este mecanismo como multiprocesamiento asimétrico (AMP), este es un mecanismo simple, ya que solo un CPU accede a las estructuras del sistema, reduciendo la necesidad de compartir datos. El segundo método consiste en SMP (simetric multiprocessing), en el que cada CPU se planifica por sí mismo; se puede usar una cola de preparados comunitaria o cada uno de los CPU pude tener una propia privada. Independientemente de eso la planificación se lleva a cabo haciendo que el planificador de cada CPU examine la cola de preparados y seleccione un proceso para ejecutarlo, mediante algún mecanismo que asegure que dos CPU no elegirán el mismo proceso y que no se perderán procesos de la cola.

Afinidad al procesador:
Como cada CPU tiene su propia cache y ahí se almacenan los datos más utilizados de la memoria principal, de cada proceso que se ejecuta en dicho CPU, en la mayoría de los SMP, se intenta evitar la migración de procesos de una CPU a otra, tratando de mantener la ejecución de un proceso en un mismo procesador. Esta afinidad CPU-proceso (que en él se está ejecutando), se denomina "Afinidad al Procesador". Sin embargo siempre existe la posibilidad de que un proceso migre del CPU donde está ejecutando a otro, entonces la afinidad es llamada "Suave". Existen otros sistemas en los que se le otorgan llamadas al sistema para especificar que un proceso no pueda bajo ninguna circunstancia migrar a otro CPU, lo que se denomina afinidad "Dura".

Equilibrado de carga:
En los sistemas SMP es importante equilibrar la carga de trabajo entre los diferentes CPU's para que ninguno este muy ocupado mientras otro está libre, pudiendo así aprovechar la potencia de tener múltiples CPU. Es importante recalcar que no en cualquier tipo de SMP es fundamental emplear un mecanismo de equilibrado de carga para distribuir equitativamente el procesamiento solamente en los casos en donde cada CPU tenga asociada una cola de preparados, dado que en los casos en los cuales se tenga una cola de preparados para todos los CPU's, cuando alguno de ellos no tiene tarea alguna, simplemente saca un proceso de dicha cola, pero en la mayoría de los sistemas operativos actuales es el de una cola de preparados por cada CPU. Los métodos para equilibrar la carga son:

- ✓ Migración comandada: existe una tarea encargada de monitorear periódicamente la distribución de trabajos en los distintos CPU's, y si se encuentra un desequilibrio migra procesos de CPU con mayores cargas de trabajo a CPU's con menores cargas de trabajo.
- ✓ Migración solicitada: los CPU's que inactivos por si solos extraen de los CPU's con mas carga de trabajo procesos para ejecutar que estuviesen en espera.

Estos dos mecanismos no tienen porque ser mutuamente excluyentes, a menudo se implementan en paralelo, como sucede en Linux, en el que cada 200 milisegundos se rechaza una migración comandada o cada vez que un CPU tiene su cola de preparados vacía realiza una migración solicitada, como método de ejecución de algoritmos de equilibrio de carga. A menudo pasa que estos mecanismos de equilibrio de carga perjudican la afinidad del procesador.

Mecanismos multihebra simétricos:
Los SMP permiten ejecutar múltiples hebras en forma concurrente, una estrategia llamada SMT (simetric multithreading) o HT (hyper threading) en los CPU Intel, consiste en otorgar una serie de CPU's lógicos en vez de físicos, la arquitectura le hace ver al sistema operativo múltiples de ellos en un mismo CPU físico a pesar de que este sea uno solo. Cada CPU lógico tiene sus propios registros de propósito general, y de estado de máquina, además cada uno es responsable de su propio tratamiento de interrupciones, por lo demás los CPU's lógicos comparten los recursos del CPU físico, como la memoria cache y los buses.

Planificaciones de hebras:

En los sistemas operativos que permiten su uso se planifican hebras del nivel kernel no procesos.

Ámbito de contienda del proceso: cuando la competición de las hebras por la CPU tiene lugar entre las hebras que pertenecen al mismo proceso, estas hebras son ULT.
Ámbito de contienda del sistema: cuando la competición de las hebras por la CPU tiene lugar entre todas las hebras del sistema, estas hebras son KLT.

Evaluación de algoritmos:
Para emplear las medidas antes mencionadas, de análisis de algoritmos de planificación, debemos definir la importancia relativa de estas medidas. A continuación mencionamos algunos de los métodos que se utilizan para ese propósito:

- ✓ Modelado determinista:
  Un método importante de evaluación es la evaluación analítica, esta toma un algoritmo específico y la carga de trabajo del sistema para generar una formula o numero que evalué el rendimiento del algoritmo para dicha caso. Uno de esos tipos de evaluación analítica es el modelo determinista, el cual toma una carga de trabajo determinada concreta, y define el rendimiento de cada algoritmo para dicha carga de trabajo.
- ✓ Modelo de cola:
  En muchos sistemas, los procesos que se ejecutan varían de un día a otro, por lo que no existe un conjunto estático de procesos (o tiempos) que se pueda emplear en el modelado determinista. Sin embargo pueden determinarse las distribuciones de ráfagas de CPU y de E/S, estas pueden medirse y luego aproximarse o simplemente estimarse. En este marco se define el análisis de redes de colas, como un método en el cual se tiene una red de servidores, cada servidor con una cola de procesos en espera, la CPU es un servidor con su cola de preparados al igual que el sistema de E/S con sus colas de dispositivos, conociendo las tasas de llegada y el tiempo de servicio, podemos calcular la utilización, la longitud media de las colas, el tiempo medio de espera, etc.
- ✓ Simulaciones:
  Ejecutar simulaciones requiere programar un modelo del sistema informático, los componentes principales del sistema se representan mediante estructura de software.
- ✓ Implementación:
  El mejor método para evaluar un algoritmo de planificación es la codificación y prueba del mismo, incluyéndolo en un sistema operativo y viendo cómo funciona. Este método posee desventajas tales como los costes, codificar el algoritmo, modificar el sistema para que lo soporte, tener en cuenta la reacción del usuario. Otra dificultad consiste en que el entorno de prueba cambiara no solo de forma usual.

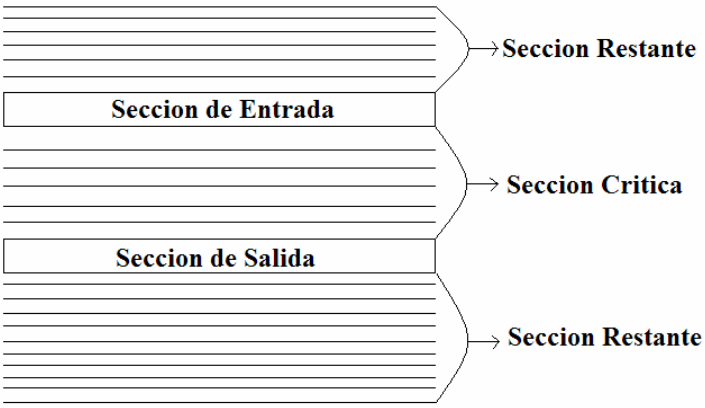
Capitulo 6:
Sincronización de Procesos

Condición de Carrera: decimos que existe si existen dos o más procesos o hebras que acceden a los mismos datos (variables) y el resultado de su ejecución dependerá del orden en que se produzcan dichos accesos (quien llega primero a los datos, y otras situaciones donde los procesos o hebras "compiten" por los accesos a los datos).

Para impedir esta "Condición de Carrera" entre procesos o hebras, se debe de alguna forma lograr "sincronizarlos" para que siempre que se acceda a los datos de a un proceso o hebra a la vez.

Sección Crítica: porción de código en la cual un proceso o hebra cooperativa modifica el o los valores de alguno o algunos de los recursos compartidos con otros procesos o hebras.

La idea consiste en que dos o más procesos o hebras no ejecuten al mismo tiempo su sección crítica. Cada proceso en un protocolo de sincronización poseerá una sección de entrada donde solicitara permiso para entrar en su sección critica, la sección critica propiamente dicha, y una sección de salida, el esquema de secciones sería el siguiente:



El resto del código es llamado sección restante como muestra la figura. Cualquier solución al problema de la sección crítica deberá cumplir los siguientes requisitos:

- ✓ Exclusión mutua: solo un proceso a la vez puede ejecutar su sección crítica.
- ✓ Progreso: si no hay procesos ejecutando su sección crítica, serán candidatos a ejecutar sus secciones críticas aquellos procesos que no estén ejecutando sus secciones restantes y la decisión de cual no podrá demorar eternamente.
- ✓ Espera limitada: existe un límite máximo de veces que un mismo proceso no logra ejecutar su sección crítica frente a otros.

Existen también múltiples secciones criticas en el código del kernel de un sistema operativo, y como puede haber múltiples procesos en modo kernel al mismo tiempo este debe valerse de algún tipo de protocolo como el anterior para no entrar en condición de carrera; algunos de estos recursos del sistema operativo son, la Tabla Global de Archivos Abiertos, o las que controlan la memoria principal, etc. Existen dos formas de implementar un kernel para solucionar la condición de carrera:

Kernel no expropiativos: aquellos en los cuales solo puede haber un proceso a la vez ejecutando código kernel (en modo kernel) y nadie puede "expropiarle" ese derecho al proceso solo él puede cederlo (por el motivo que fuere) a otro, este tipo esta esencialmente exento de condiciones de carrera.
Kernel expropiativo: diferente del tipo anterior, un proceso ejecutando en modo kernel en un esquema de este tipo puede ser desalojado por otro que solicita su derecho, por lo tanto este tipo de sistemas operativos deben ser diseñados cuidadosamente de forma tal que protejan los recursos compartidos del sistema.

Hardware de sincronización:
Para solucionar las cuestiones como la condición de carrera hay una herramienta en particular denominada cerrojo que permite inhabilitar uno o más procesos cuando otro (único) se encuentra en su sección critica, esto ocurre mediante dos operaciones aplicables al cerrojo, la adquisición y liberación del mismo, que se ubican en la sección de entrada y de salida respectivamente. La implementación del mismo se realiza mediante hardware o API's de software, todas ellas se implementan mediante las premisas de bloqueo. Una posible solución sería evitar las interrupciones cuando se modifica una variable por parte del exterior hacia el CPU, si este fuera único, que es el método que de hecho utilizan los kernels no expropiativos. Sin embargo en entornos multiprocesador no sería aplicable este tipo de solución, dado que bloquear las interrupciones tomaría demasiado tiempo, entonces se opta por otro tipo de solución consistente en una instrucción de hardware que permita consultar o modificar el contenido de una palabra o intercambiar los contenidos de dos o más palabras automáticamente, es decir como unidad de trabajo ininterrumpible. Las dos principales se pueden "representar" en un lenguaje de alto nivel como:

```
Boolean TestAndSet( Boolean *Target)
{
        Boolean RV = *Target;
        *Target = TRUE;
        return RV;
}
```

La instrucción anterior se ejecuta de forma atómica, y toma el valor de "Target" y lo modifica a "TRUE", y devuelve su valor anterior.

```
void Swap( Boolean *a, Boolean *b)
{
        Boolean TEMP = *a;
        *a = *b;
        *b = TEMP;
}
```

La instrucción anterior también se ejecuta de forma atómica, e intercambia los valores de "a" y "b". Las instrucciones "TestAndSet" y "Swap", no cumplen con los tres principios del protocolo de bloqueo por sí mismas, deberán ser implementadas añadiendo código extra de maneara de cumplirlo.

Semáforos:
Dado que las soluciones de hardware mencionadas anteriormente no son triviales para el programador de aplicaciones, se define por lo general (caso aplicaciones) una estructura denominada “Semáforo”, a la cual se le pueden aplicar dos operaciones Wait(), y Signal(). Esta estructura, y las operaciones que le son aplicables, se pueden definir en términos de un lenguaje de alto nivel como:

```
typedef struct semáforo
{
	int valor;
	struct PCB *lista;
}semáforo;

void Wait( semáforo *S)
{
	(*S).valor--;
	if((*S).valor<0)
	{
		encolarproceso(&(*S).lista);
		Block();
	}
}

void Singal( semáforo *S)
{
	struct PCB auxPCB;

	if((*S).valor<0)
	{
		auxPCB = desencolarproceso(&(*S).lista);
		Wakeup(auxPCB);
	}
	(*S).valor++;
}
```

Cuando se realiza la operación Wait() sobre el semáforo, el proceso se auto bloquea, y como a cada semáforo se le asocia una cola de procesos bloqueados (cola de PCB’s), se almacena a este último proceso allí, con el valor de “estado” de su PCB en “Wait”, y a continuación se transfiere el control al planificador de CPU. Cuando se ejecuta la operación Signal(), se levanta al primer proceso de la cola de procesos bloqueados (la primer PCB), asociados al semáforo, y se cambia el estado de su PCB a “Ready”, y se lo ubica en la cola de Ready. Las llamadas al sistema Block() y Wakeup(), son las encargadas de bloquear y desbloquear un proceso respectivamente. Cabe aclarar que los semáforos se clasifican en dos grandes tipos:

- ✓ Semáforos contadores: no tienen límite en cuando a su valor.
- ✓ Semáforos binarios: solo pueden ser “1” o “0” (también llamados mutex).

Nuevamente en un sistema de un único CPU es fácil conservar las operaciones Wait() y Singal() atómicas, dado que se inhabilitan las interrupciones y por ende ninguno otro

proceso podría “expropiarle”, aunque sea por esa función, el CPU a la aplicación. En sistemas de múltiples CPU, deshabilitar las interrupciones por cada CPU, como dijimos antes no es posible, por el considerable tiempo que toma, y la baja en el rendimiento del sistema, por lo tanto se deberán proporcionar técnicas alternativas (como TestAndSet, Swap, etc.) para asegurar la atomicidad de Wait() y Singal().

Capitulo 7:
Interbloqueos

Modelo de sistema:
Un sistema consta de un número finito de recursos que se distribuyen entre una serie de procesos (en competencia por los mismos). Los recursos se clasifican en tipos y estos tienen un número finito de instancias, si un proceso solicita una de ellas, la asignación de cualquiera de ellas al mismo satisfará la solicitud. La secuencia de empleo de un recurso por parte de un proceso es la siguiente:

- ✓ Solicitud: si no se puede conceder de inmediato el proceso espera hasta que pueda adquirir el recurso.
- ✓ Uso: el proceso puede operar sobre el recurso.
- ✓ Liberación: el proceso libera el recurso.

La solicitud y liberación de un recurso por parte de un proceso se traduce en una system call (Request();, Rlease(); - Dispositivos; Open();, Close(); - Archivos; Allocate();, Free(); - Memoria). Las solicitudes y liberaciones de recursos que el sistema operativo no gestiona, pueden realizarse mediante Wait(); y Signal(); de semáforos. El sistema operativo maneja una tabla de recursos en la que se registra de cada recursos si esta libre o asignado, y en este último caso a que proceso está asignado actualmente Si un proceso solicita un recurso que en ese momento está asignado, puede añadirse le a la cola de procesos en espera de dicho recurso

Interbloqueo:
Un conjunto de procesos se hallaran en interbloqueo si todos los procesos del conjunto están esperando a que se produzca un suceso que solo puede darse como resultado de la actividad de otro proceso del conjunto.

Caracterización de los interbloqueos:
En un interbloqueo, como se intuye de su definición, los procesos del conjunto nunca terminan de ejecutarse, y los recursos del sistema están ocupados.

Condiciones necesarias para que haya interbloqueo:

- ✓ Exclusión mutua: al menos un recurso debe estar en modo no compartido, es decir solo puede usarlo un proceso a la vez y el resto de los procesos que lo soliciten tendrán que esperar hasta que este sea liberado.
- ✓ Retención y espera: un proceso debe estar reteniendo al menos un recurso y esperado adquirir al menos otro recurso que actualmente retiene o retienen otros procesos.
- ✓ Sin desalojo: los recursos no pueden ser desalojados, es decir que un recurso solamente puede ser liberado voluntariamente por el proceso que lo retiene.
- ✓ Espera circular: debe existir un conjunto de procesos $C=\{P_0, P_1, P_2,\ldots, P_n\}$ tal que "P_0" espera el recurso que retiene "P_1", "P_1" espera el que retiene "P_2" y así hasta llegar a "P_n" que espera el que retiene "P_0".

Grafo de asignación de recursos:
Existe una forma más precisa de definir un interbloqueo, y esa conociste en un grafo dirigido denominado "Grafo de asignación de recursos del sistema". Este grafo está formado por:

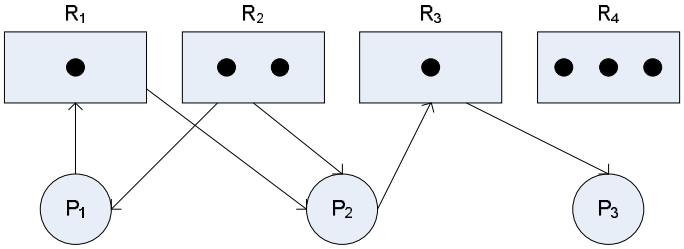
Vértices:
- $P=\{P_0, P_1, P_2,..., P_n\}$ (Procesos activos del sistema).
- $R=\{R_0, R_1, R_2,..., R_n\}$ (Todos los tipos de recursos del sistema).

Aristas:
- $P_i \longrightarrow R_j$ (Arista de solicitud: el proceso "P_i" ha solicitado una instancia del tipo de recurso "R_j" y espera por la misma).
- $P_i \longleftarrow R_j$ (Arista de asignación: se ha asignado al proceso "P_i" una instancia del tipo de recurso "R_j").

De los vértices los procesos se representan mediante círculos y los tipos de recursos mediante rectángulos, habiendo dentro de estos últimos un punto por cada instancia de ese tipo de recurso en el sistema. Cuando un proceso "P_i" solicita una instancia de una tipo de recurso "R_j", se dibuja una arista de solicitud entre "P_i" y "R_j", si se concede la solicitud esta arista se transforma instantáneamente en arista de asignación, cuando el proceso ya no necesita acceder al recurso, este se libera y la arista de asignación se borra. A modo de ejemplificación:

$P=\{P_1, P_2, P_3\}$;
$R=\{R_1,R_2,R_3,R_4\}$;
$E=\{P_1 \longrightarrow R_1, P_2 \longrightarrow R_3, R_1 \longrightarrow P_2, R_2 \longrightarrow P_2, R_2 \longrightarrow P_1, R_3 \longrightarrow P_3\}$;
R_1: Una instancia.
R_2: Dos instancias.
R_3: Una instancia.
R_4: Tres instancias.

Entonces:



Se puede demostrar, que si no hay ciclos en el grafo, entonces no hay interbloqueo, pero de tener alguno puede existir interbloqueo o no. En el caso particular en el que cada tipo de recurso implicado en el ciclo encontrado tenga solamente una instancia entonces la existencia de dicho ciclo implica interbloqueo.

Métodos para tratar interbloqueos:
Existen tres alternativas para tratar con los interbloqueos:

- ✓ Utilizar un protocolo para impedirlos o evitarlos, asegurando que el sistema nunca entre en el estado de interbloqueo.
- ✓ Permitir que el sistema entre en estado de interbloqueo, detectarlo y realizar una recuperación
- ✓ Ignorar el problema y actuar como si no existiese tal cosa como un interbloqueo.

La tercera alternativa, es la más utilizada por los sistemas operativos, y deja a cargo del usuario la responsabilidad de no generar interbloqueos.
Prevención de interbloqueos:
Asegurando que no se cumple cualquiera de las cuatro condiciones necesarias para que se produzca interbloqueo podemos prevenir la aparición de interbloqueos. Para que no se cumpla:

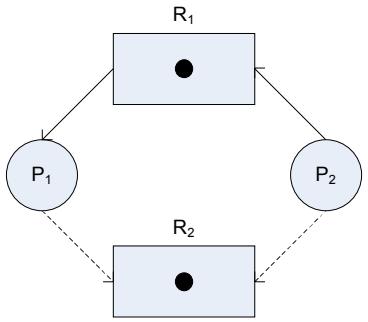
- ✓ Mutua exclusión: aquellos recursos que si pueden compartirse no sufren de mutua exclusión, porque el acceso a ellos no es mutuamente excluyente sino que se pueden compartir. Sin embargo en general no es práctico negar esta condición dado que hay muchos recursos que son esencialmente no compartibles.
- ✓ Retención y espera: se deberá garantizar que cuando un proceso solicite un recurso, no este reteniendo ningún otro recurso. Una forma de implementarlo seria exigir a cada proceso que solicite todos sus recursos antes de comenzar su ejecución, esto se puede realizar requiriendo que las llamadas al sistema que soliciten los recursos para un proceso precedan a todas las demás al sistema. Otra alternativa seria que un proceso pudiera solicitar algunos recursos y utilizarlos, pero antes de solicitar cualquier otro adicional tuviese que liberar todos los recursos que tuviese asignados en ese momento. Las desventajas de ambos protocolos consisten en que la tasa de utilización de los recursos pueden ser baja, y la alta inanición por solicitud de recursos sumamente requeridos.
- ✓ Sin desalojo: para impedir que se cumpla que no se pueden desalojar un recurso asignado a un proceso, se podría desalojar los recursos asignados a un proceso determinado cuando este solicite un recurso que no se le puede asignar de forma inmediata, los recursos desalojados se añaden a la lista de recursos que espera dicho proceso, el proceso no reiniciara sino hasta que se le devuelvan sus recursos antiguos junto con los nuevos que está solicitando. Otra alternativa conociste en que si un proceso solicita recursos que no se le puede asignar de forma inmediata, pero que están retenidos por otro proceso actualmente en espera de otros recursos, se le pueden desalojar dichos recurso al proceso en espera, y asignárselos al que los solicita, y de no haber un proceso en espera reteniendo los recursos necesarios, el solicitante deberá esperar a que se le asignen, sometiéndose a la posibilidad de que le desalojen recursos durante la espera, el proceso solo podrá reiniciarse cuando se le asignen los nuevos recursos que ha solicitado y se le devuelvan aquellos que se le hayan desalojado mientras esperaba.
- ✓ Espera circular: para impedir la espera circular se puede definir una función $F:R \rightarrow N$, que dado un tipo de recurso "R_j" devuelve un numero natural univoco correspondiente a dicho tipo de recurso. Una vez definida la función que deberá respetar criterios como el orden normal de utilización de los recursos en un sistema, se podría permitir a un proceso solicitar inicialmente cualquier numero

de instancias de un cierto tipo de recurso "R_i", pero a partir de ella dada una nueva solicitud de otro tipo de recurso "R_j", esta será válida si y solo si "$F(R_j) > F(R_i)$". Alternativamente al protocolo antes mencionado, se podría requerir que si un proceso solicita instancias del tipo de recurso "R_j", para que la solicitud sea válida, este deberá liberar primero cualquier recurso "R_i" tal que "$F(R_j) \leq F(R_i)$". El respeto por la ordenación o la jerarquía en los programas es responsabilidad del programador de aplicaciones.

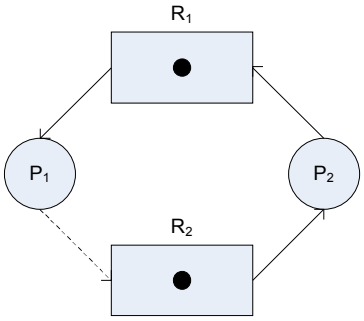
Evasión de interbloqueos:
Los algoritmos de prevención nombrados en la sección anterior acarrean la desventaja de la producción de posibles efectos colaterales, como baja tase de utilización de los dispositivos y un menor rendimiento del sistema. Un método alternativo consiste en requerir información adicional sobre cómo van a ser solicitados los recursos y como van a ser liberados por porte de un proceso, esto le permite al sistema determinar si el proceso debe esperar o no, con el fin de prevenir un futuro interbloqueo. Existen pero la más simple y útil requiere que cada procesos declare el número máximo de recursos de cada tipo que puede necesitar.

Estado seguro:
Si existe una "Secuencia segura" el sistema está en estado seguro. Una "Secuencia segura", es una secuencia de procesos $<P_0, P_1, P_2, ..., P_n>$ en la cual para cada proceso "P_i", las solicitudes de recursos que pueda todavía hacer el mismo, puedan ser satisfechas mediante los recursos actualmente disponibles, junto con los recursos retenidos por todos los procesos "P_j" pertenecientes a la secuencia, con $j<i$. Si no existe tal secuencia se dice que el estado del sistema es inseguro. Un estado seguro implica que no puede producirse interbloqueo, por ende el estado deber ser inseguro para que se produzca interbloqueo, sin embargo no todos los estados inseguros producen interbloqueo.

Modificación del grafo de asignación:
Además de las aristas de solicitud y asignación ya descritas, introducimos las aristas denominadas de declaración, estas indican que un proceso puede llegar a solicitar el recurso al que está dirigida la arista en algún instante en el futuro, esta se gráfica con una flecha punteada dirigida desde el proceso hacia el recurso. Cuando el proceso solicita un recurso su arista de declaración se convierte en una arista de solicitud, cuando un proceso libera un recurso la arista de asignación del mismo se transforma en una de declaración Con este grafo podemos prevenir la producción de interbloqueos mediante la detección de los que podrían llegar a convertirse en ciclos. A modo de ejemplificación se tiene:



En este caso aunque "R_1" este libre no se lo podríamos asignar a "P_1" por ejemplo, dado que si así lo hiciéramos se crearía un ciclo hipotético que implicaría que el sistema estaría en estado inseguro, esto podría llegar a desatar un interbloqueo, si "P_0" decidiera convertir su arista de declaración en arista de solicitud, esta situación hipotética se ilustra de la siguiente manera:



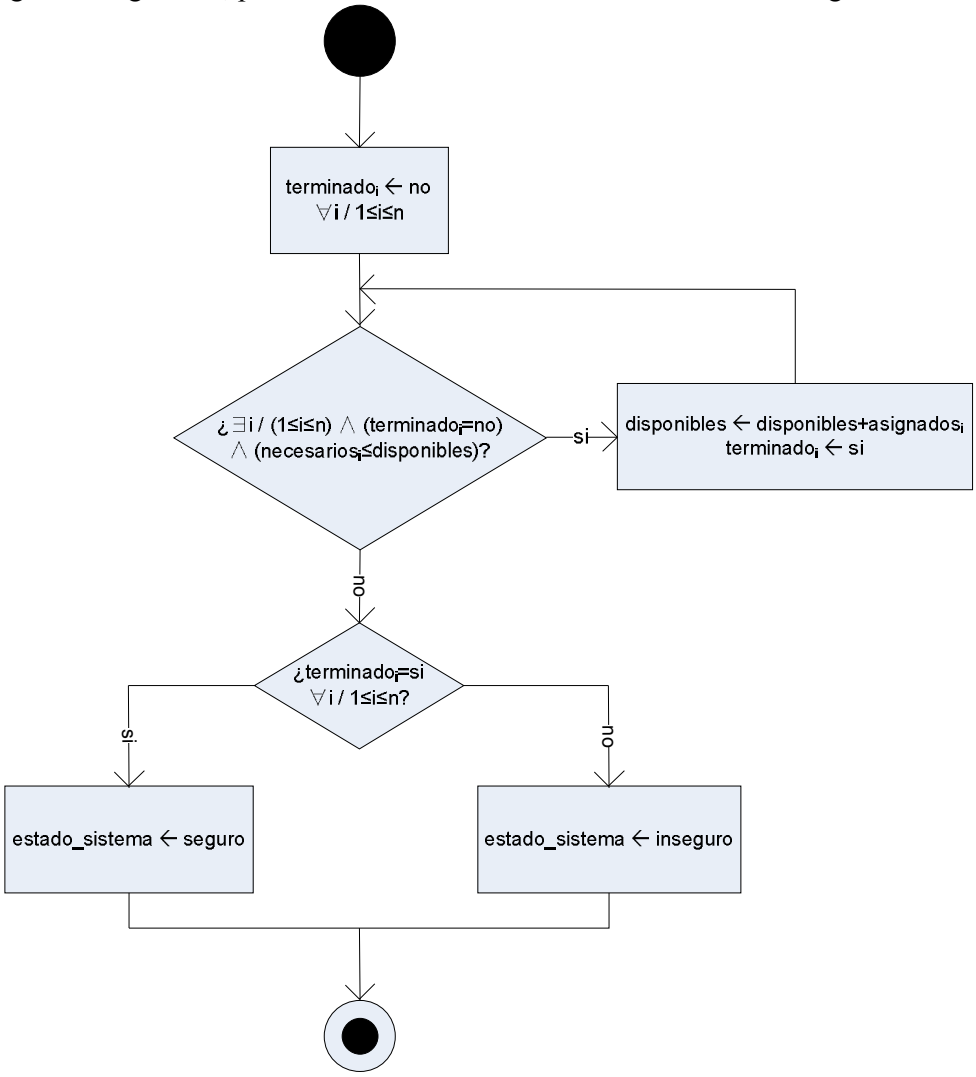
Algoritmo del banquero:

El esquema anterior no es aplicable a los sistemas con múltiples instancias de cada tipo de recurso, es por dicha razón que propondremos un algoritmo menos eficiente pero que se puede utilizar en sistemas donde halla as de una instancia en algún recurso, el cual se denomina "Algoritmo del banquero". En este algoritmo cuando un proceso nuevo ingresa al sistema debe declarar el número máximo de instancias de cada tipo de recurso que puede necesitar, este valor no puede exceder el total de recursos del sistema. Cuando el sistema asigna recursos debe controlar que la asignación no desemboque en un estado inseguro, en caso afirmativo los recursos no se asignan, y el proceso deberá esperar hasta que los otros procesos liberen los suficientes recursos, si el caso es el contrario se podrán asignar los recursos. Las estructuras de datos y los datos elementales, implicados en el "Algoritmo del banquero" son:

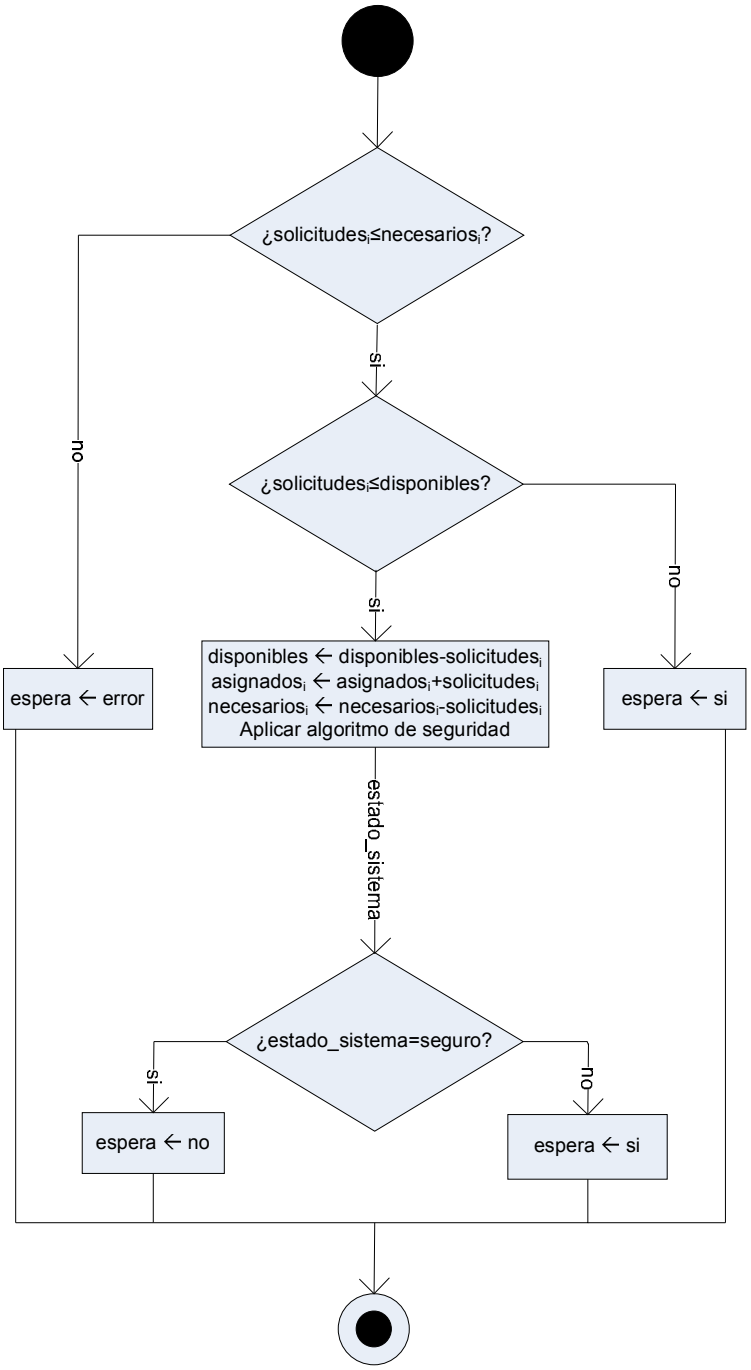
- ✓ m: cantidad de tipos de recursos distintos en el sistema.
- ✓ n: cantidad de procesos activos distintos en el sistema.
- ✓ Disponibles: es un vector de "m" componentes que indica el número de recursos disponibles de cada tipo de recurso, en el cual el valor de "$Disponibles_i$" es la cantidad de instancias disponibles del "i-esimo" tipo de recurso.
- ✓ Máximos: es una matriz de "n x m" elementos que indica la demanda máxima de cada proceso, en la cual el valor de "$Máximos_{ij}$" es la cantidad máxima de instancias del "j-esimo" tipo de recurso que puede solicitar el "i-esimo" proceso.
- ✓ Asignados: es una matriz de "n x m" elementos que indica el numero de recursos de cada tipo actualmente asignados a cada proceso, en la cual el valor de "$Asignados_{ij}$" es la cantidad de instancias del "j-esimo" tipo de recurso que tiene asignadas actualmente el "i-esimo" proceso.
- ✓ Necesarios: es una matriz de "n x m" elementos que indica la necesidad restante de recursos de cada proceso, en la cual el valor de "$Necesarios_{ij}$" es la cantidad de instancias del "j-esimo" tipo de recurso que podría llegar a necesitar el "i-esimo" proceso para completar sus tareas. Observamos que se puede obtener mediante la resta de la matriz de "Máximos" con la de "Asignados".

Estas estructuras de datos varían con el tiempo, tanto en tamaño como en valor. Para simplificar la presentación del "Algoritmo del banquero", redefinimos el operador matemático "≤"; sean dos vectores cuales se quieran "A" y "B" de longitud igual "x" ambos, decimos que "$A \leq B$" si y solo si "$A_i \leq B_i$" $\forall i / 1 \leq i \leq x$.

Algoritmo de seguridad:
El siguiente algoritmo, permite saber si el estado actual del sistema es seguro o no:



Algoritmo de solicitud de recursos:
El siguiente algoritmo permite determinar si las solicitudes pueden concederse de forma segura:

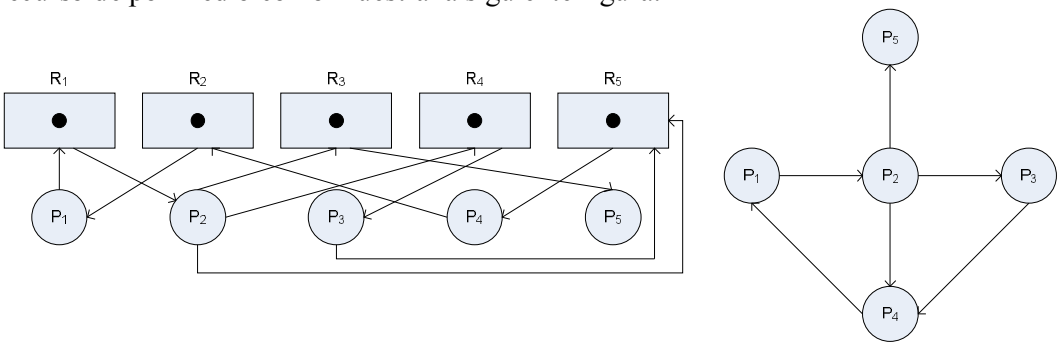


Detección de interbloqueos:
Como habíamos enunciado al principio, existe la variante de permitir que pueda producirse interbloqueo, en este caso deberá pasar que el sistema proporcione:

- ✓ Un algoritmo que examine el estado del sistema y detecte si se ha producido interbloqueo.
- ✓ Un algoritmo para recuperarse del interbloqueo.

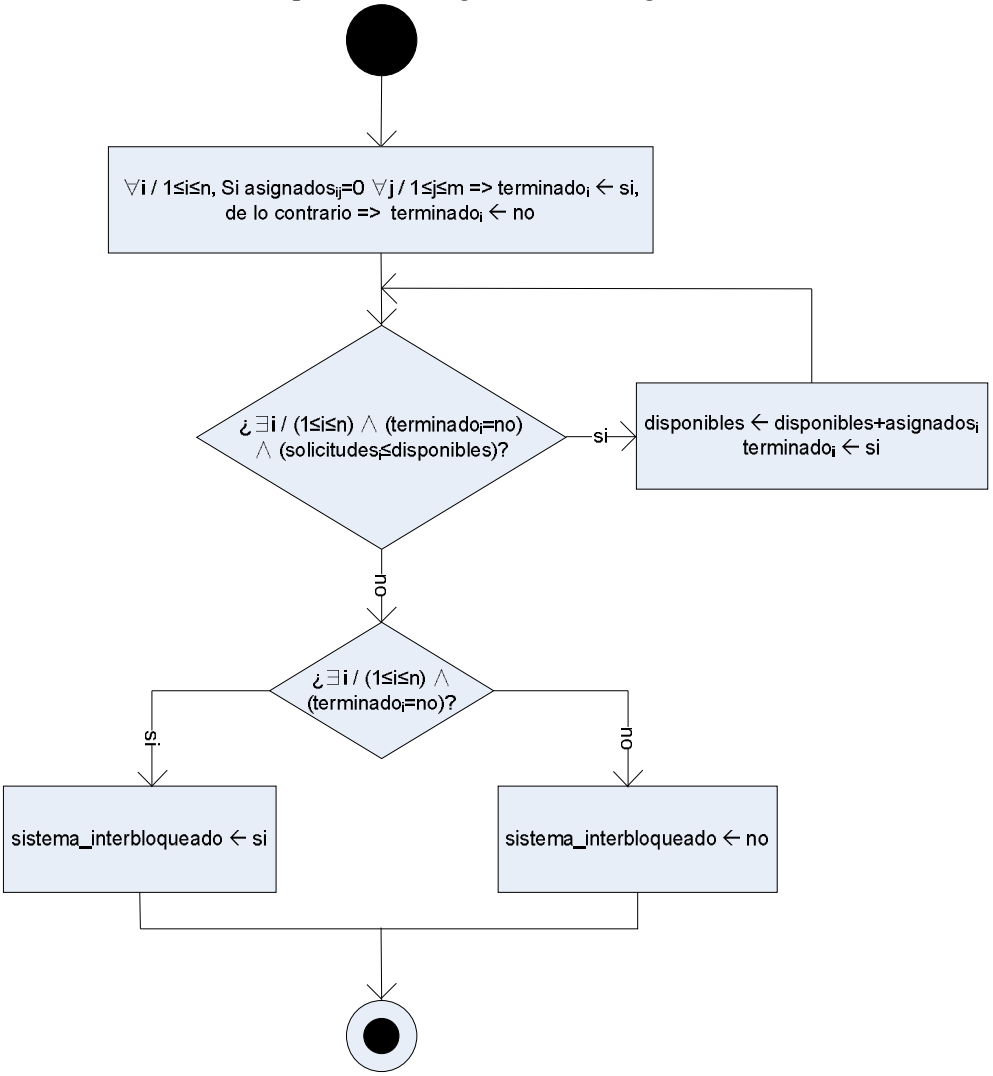
Esta alternativa no solo acarrea un conste en tiempo de ejecución en cuanto a la detección y recuperación del sistema, sino también las potenciales perdidas inherentes al proceso de recuperación de un interbloqueo.

Una sola instancia de cada tipo de recurso:
En este tipo de casos el algoritmo de detección es una variante del mencionado "Grafo de asignación de recursos", llamada "Grafo de espera", el cual no tiene nodos de recursos, y los de procesos se apuntan entre sí indicando si un proceso apunta a otro, que el proceso que apunta está esperando que el proceso apuntado libere los recursos que necesita. Esto se logra sintetizando el "Grafo de asignación de recursos" convirtiendo los casos donde un proceso apunta a un recurso y ese recurso a otro proceso, en una sola arista que apunte del primer proceso al segundo eliminando el recurso de por medio como muestra la siguiente figura:



Si el "Grafo de espera" contiene un ciclo implicara que se produjo interbloqueo, es por esto que en este esquema es importante que se controle mediante un algoritmo paulatinamente si existe un ciclo en el grafo de espera.

Varias instancias de cada tipo de recurso:
El algoritmo que se emplea para este tipo de situación posee las mismas estructuras que el “Algoritmo del banquero”, solo que sin la matriz de “Máximos” y la de “Necesarios”, y con una matriz llamada de “Solicitudes”, cuya cantidad de elementos es “n x m”, y que indica la solicitud actual efectuada por cada proceso, en esta matriz, el valor de “$Solicitudes_{ij}$” es la cantidad de instancias del “j-esimo” tipo de recurso que está solicitando el “i-esimo” proceso. El algoritmo es el siguiente:



Utilización del algoritmo de detección:
Se debe considerar con qué frecuencia se producen interbloqueos y cuantos procesos se verán afectados cuando se produzca uno. Si los interbloqueos se producen frecuentemente entonces el algoritmo de detección debe involucrarse frecuentemente. Otra alternativa dado que los interbloqueos se producen cuando a un proceso no se le pueden asignar inmediatamente los recursos, es invocarlo cada vez que este suceso acontezca. También se podría invocar el algoritmo en un lapso de tiempo terminado, o cuando la utilización del CPU caiga en un porcentaje.

Recuperación de un interbloqueo:
Se puede informar al operador que se ha producido un interbloqueo, otra opción es que el sistema haga automáticamente la recuperación del interbloqueo. A esta última

alternativa existen dos formas de implementarla, una interrumpiendo uno o más procesos, para romper la espera circular, la otra desalojando algunos recursos de uno o más de los procesos bloqueados.

Terminación de procesos:
Para eliminar interbloqueos interrumpiendo un proceso, se utilizan dos métodos los cuales implican que el sistema reclama todos los recursos asignados a los procesos terminados:

- ✓ Interrumpir todos los procesos interbloqueados: esto terminara con e interbloqueo, pero a un costo muy alto; el de perder todo el trabajo realizado por los procesos.
- ✓ Interrumpir un proceso cada vez hasta que el sistema no se encuentre mas en interbloqueo, esto conlleva un costo en cuanto a trabajo, dado que por cada proceso interrumpido se deberá invocar el algoritmo de detección de interbloqueos para determinar si todavía hay interbloqueo.

Si optamos por interrumpir un grupo de procesos deberemos elegir aquellos que sean poco costosos en su terminación, los factores que determinan que sean económicos son:

- ✓ La prioridad del proceso.
- ✓ Cuanto se ejecuto el proceso, y cuanto le queda por ejecutar.
- ✓ Cuantos, y que tipos de recursos ha utilizado el proceso.
- ✓ Cuantos más recursos necesita para completarse.
- ✓ Cuantos procesos harán falta terminar.
- ✓ Si se trata de un proceso interactivo o de procesamiento por lotes.

Apropiación de recursos:
El método consiste en desalojar recursos de una serie de procesos y asignarlos a otros procesos hasta que el ciclo de interbloqueo se interrumpa, hay tres cuestiones a tener en cuenta:

- ✓ Selección de una víctima: decidir de qué proceso extraer los recursos y cuales, es necesario determinar el orden de apropiación de modo de minimizar costos.
- ✓ Anulación: que se debe hacer con los procesos que se han seleccionado para desalojar sus recursos, se deben reiniciar a un estado seguro y reiniciarlos a partir de ese estado, pero como por lo general no se puede determinar un estado anterior de seguridad, se opta por anular completamente dichos procesos y reiniciarlos.
- ✓ Inanición: como se puede garantizar que no se le desalojen los recursos siempre al mismos proceso, se debe asegurar que un proceso solo puede ser seleccionado como víctima un pequeño número finito de veces, un alternativa es incluir el numero de anulaciones efectuadas dentro del algoritmo de cálculo de coste.

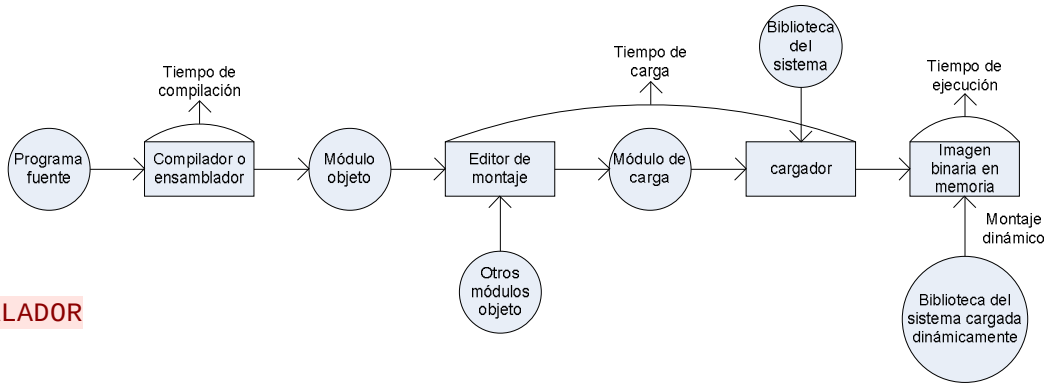
Capitulo 8:
Memoria Principal

En este capítulo analizaremos diversas formas de gestionar la memoria.

Fundamentos:
La memoria está compuesta de una gran matriz de palabras o bytes, cada una con su propia dirección La CPU extrae instrucciones de la memoria de acuerdo con el valor del contador de programa. Estas instrucciones pueden provocar operaciones adicionales de carga o de almacenamiento en direcciones específicas de memoria.

Hardware básico:
Los registros integrados dentro el CPU y la memoria son las únicas áreas de almacenamiento a las que la CPU puede acceder directamente, lo que implica que todas las instrucciones en ejecución, y los datos utilizados por esas instrucciones, deberán encontrarse almacenados en algunos de estos dispositivos de acceso directo, de no encontrarse deberán llevarse hasta allí antes de que la CPU pueda operar con ellos. Se puede acceder a los registros de la CPU en un único ciclo de reloj, distinto del acceso a la memoria, que puede tomar muchos ciclos de reloj, durante los cuales el CPU necesitara normalmente detenerse, ya que no dispondrá de los datos requeridos para completar las instrucciones que está ejecutando. Dado el importante impacto que esto tiene en el rendimiento, se debe encontrar una solución al problema, la cual implica añadir una memoria rápida entre el CPU y la memoria, llamada cache. También se debe garantizar una correcta operación que proteja al sistema operativo de los accesos de los procesos de usuarios y también a unos procesos de usuarios de otros, esta protección la garantiza el hardware y puede implementarse de varias formas. Una posible protección sería la de asignar a un proceso un rango de direcciones legales a las cuales únicamente dicho proceso pueda acceder. Para implementar este mecanismo dicha sección podrá estar delimitada por dos registros, uno llamado “Base” y otro “Limite”. El primero almacena la dirección de memoria más pequeña de la sección, y el segundo el tamaño de rango de la sección Para realizar la protección el hardware de la CPU compara todas las direcciones generadas por el proceso con el valor de dichos registros, si se produce un acceso ilegal se genera una interrupción que el sistema operativo trata como error fatal. Los registros “Base” y “Limite” solo pueden ser modificados por el sistema operativo, que utiliza una instrucción privilegiada especial, que solo puede ser modificada en modo kernel y, como solo el sistema operativo puede trabajar en ese modo, de ahí que el único que los puede modificar es el. El sistema operativo puede también acceder al espacio de memoria de los procesos de usuario, así como al suyo.

Reasignación de direcciones:
Los programas, como ya sabemos, residen en disco en forma de archivos ejecutables, estos al momento de la ejecución deberán ser cargados en memoria y ser colocados dentro de un proceso. Una vez en ejecución, dependiendo del mecanismo de gestión de la memoria, pueden pasar de esta al disco y viceversa. Los procesos del disco que esperan a ser cargados en memoria para su ejecución, forman una cola llamada de entrada. En la mayoría de los casos el programa de usuario tendrá que pasar por varias etapas antes de ser ejecutado, como esquematiza la siguiente figura:



SEÑALADOR

La reasignación de las instrucciones y datos a direcciones de memoria puede realizarse en cualquiera de las tres etapas marcadas en la figura anterior, estas son:

- ✓ Tiempo de compilación: si sabemos donde residirá el proceso al momento de compilarlo, se habla de generar un "Código absoluto". Este esquema es bastante rígido, dado que si se quisiese reubicar el proceso en otra posición, se deberá recompilarlo. Un buen ejemplo de este tipo de programas son los ".com" de MS-DOS.
- ✓ Tiempo de cargar: si no sabemos donde residirá el proceso en tiempo de compilación el compilador ha de generar un "Código reubicable", retardando la reasignación hasta el momento de carga. De querer reubicar el proceso simplemente se vuelve a cargar el mismo.
- ✓ Tiempo de ejecución: si el proceso puede reubicarse de una porción de memoria a otra durante su ejecución, entonces es necesario retardar la reasignación hasta el instante de ejecución Para este esquema será necesario un hardware determinado este esquema es el que utilizan la mayoría de los sistema operativos.

Espacio de direcciones lógico y físico:
Antes de comenzar esta sección es necesario aclarar algunos conceptos:

- ✓ Dirección lógica de memoria: una dirección generada por la CPU.
- ✓ Espacio de direcciones lógicas: conjunto de todas las direcciones lógicas generadas por un programa.
- ✓ Dirección física de memoria: una dirección vista por la unidad de memoria.
- ✓ Espacio de direcciones físicas: conjunto de todas las direcciones físicas que se corresponden con el espacio de direcciones lógicas de un programa.

Cabe aclarar también que en el tercer esquema difieren los espacios de direcciones lógicas y físicas de un programa, como dijimos antes en este esquema es necesario hardware para implementarlo, y ese es específicamente la "Memory management unit", encargada de realizar la traducción de direcciones lógicas a físicas, para efectuar esta tarea existen varios métodos que enunciaremos más adelante. A modo de ejemplificación de esta operación de traducción (únicamente para darnos una idea de en que concite dicha operación) está el "Sistema de reasignación" de MS-DOS, el cual tiene un registro de "Reubicación" que es un registro "Base", que tiene donde inicia el proceso en la memoria, si a esto sumamos el valor de la dirección lógica que el usuario provea obtendremos la dirección física del mismo, lo que implica por ejemplo, que si el

usuario quiere acceder a la dirección lógica “24” que él ve como física y el registro de “Reubicación” tiene el valor “1200”, entonces la dirección física será “1224”.

Carga dinámica:
Es un esquema utilizado para obtener una mejor utilización del espacio de memoria, la idea es que una rutina no se cargue en memoria hasta que se invoque, estas se encuentran en el disco en un formato de carga reubicable. Si una rutina de un proceso solicita otra rutina, la rutina invocante verifica que la otra este cargada si no lo está invoca al “Cargador de montaje reubicable” el cual carga la rutina invocada, actualiza las tablas de direcciones del programa, y le pasa el control a la rutina recién cargada. La ventaja de este mecanismo es que las rutinas que no se utilicen no se cargaran. El mecanismo es responsabilidad de los usuarios, dado que está en ellos diseñar programas para poder aprovechar dicho método
Montaje dinámico y bibliotecas compartidas:
El llamado montaje estático concite básicamente en incluir en cada programa del sistema, una copia de su biblioteca de lenguaje, o al menos las rutinas a las que se haga referencia, dentro de su imagen ejecutable Este requisito hace que se desperdicien tanto espacio de disco como memoria. En contraposición, en el esquema de montaje dinámico, se incluye en la imagen binaria del proceso un “Stub” o un pequeño fragmento de código, que permite localizar la rutina adecuada de biblioteca residente en memoria, o como cargar la biblioteca si esa rutina no está todavía presente. El “Stub” corrobora si la rutina necesaria esta en memoria, si no está, la carga, y luego tanto si esta como si no, la sección de código de “Stub” se auto reemplaza por una referencia directa a la rutina ya cargada en memoria y la ejecuta. La ventaja principal es que todos los procesos que utilicen una determinada biblioteca de lenguaje, solo necesitan ejecutar una copia del código de la biblioteca. Puede emplearse su funcionalidad, para que una vez actualizada una biblioteca de código, todos los programas que hagan referencia a la biblioteca, empleen automáticamente la versión más reciente. Suele incluirse información de la versión de la biblioteca, a efectos de que o haya ejecuciones incompatibles, de nuevas versiones de biblioteca. Solo los programas que se compilen con la nueva versión de la biblioteca se verán afectados por los cambios incompatibles incorporados en ella, otros programas montados antes de que se instalara la nueva biblioteca, utilizaran la antigua, este mecanismo es llamado de “Bibliotecas compartidas”. El montaje dinámico, a diferencia de la carga dinámica suele requerir ayuda del sistema operativo.

Intercambio:
Un proceso debe estar en memoria para ejecutarse, pero los procesos pueden ser intercambiados temporalmente mientras no se ejecuten sacándolos de memoria y poniéndolos en un almacén de respaldo y viceversa para su ejecución Normalmente el almacén de respaldo será u disco, lo más rápido posible, y suficientemente grande como para poder albergar copias de todas las imágenes de memoria, para todos los usuarios y proporcionar un acceso directo a dichas imágenes La cola de preparados, no solo contiene procesos preparados en memoria, sino también en disco, cada vez que el planificador selecciona un proceso de esta cola y lo pasa al despachador para que lo ejecute, este verificara que el proceso este en memoria, si no es así lo carga de disco, intercambiando con otro proceso si no hay espacio en la memoria, luego recarga los requisitos y transfiere el control al proceso seleccionado. Este mecanismo tiene básicamente dos problemas:

- ✓ El tiempo necesario de cambio de contexto es relativamente alto.
- ✓ Si se quiere intercambiar un proceso, se debe estar muy seguro de que este, este completamente inactivo.

En el caso del primer problema, una solución consiste en almacenar información sobre cuanta memoria cada proceso está realmente utilizando, para no realizar transferencias entre el almacenamiento y la memoria, que es lo que más toma, de cosas que luego no se utilizaran. Para el segundo caso es necesario prestar atención a las operaciones de E/S del proceso a intercambiar, dado que si al momento de intercambiar un proceso, este está en espera de una E/S, al nuevo proceso cargado dicha operación de E/S podría intentar utilizarle su memoria, con la lógica de que el proceso anterior aun sigue en memoria, en esa posición Para solucionar este tipo de situaciones hay dos alternativas:

- ✓ No intercambiar procesos que tengan una operación de E/S pendiente.
- ✓ Ejecutar las operaciones de E/S a través de búferes del sistema operativo.

Asignación de memoria contigua:
La memoria esta usualmente particionada en dos:

- ✓ Una partición para el sistema operativo.
- ✓ Una partición para los procesos.

La primera se puede situar en las posiciones más altas o más bajas de la memoria, esta decisión depende generalmente de la posición de la "Interrupt descriptor table" (IDT), como está casi siempre se ubica en la parte baja de la memoria, también lo hará el sistema operativo en esos casos, y tomaremos esto último como convención Como uno de los objetivos del sistema operativo es la multiprogramación, tendremos que considerar como asignar la memoria a los procesos que se encuentran en la cola de entrada esperando a ser cargados en memoria. En este esquema de asignación, cada proceso está contenido en una única sección contigua de memoria.

Asignación de memoria:

- ✓ Método MFT: una forma de asignar la memoria consiste en dividirla en varias particiones de tamaño fijo, cada una de las cuales podrá albergar un proceso, de modo que el grado de multiprogramación estará limitado por la cantidad de particiones. Para este método, si esta libre una de las particiones se carga en ella un proceso de la cola de entrada, y cuando un proceso termina libera su partición asignada.
- ✓ Método MVT: es un generalización del método de particiones fijas. En este método el sistema operativo mantiene una tabla que indica que particiones de memoria están disponibles y cuales están ocupadas, inicialmente todas la memoria está disponible para los procesos de usuario y se considera como un único bloque de gran tamaño denominada agujero. A partir de ahí en cualquier momento tendremos una lista de tamaños de bloque disponibles y una cola de entrada de procesos, el sistema operativo puede decidir planificar dicha cola, para ir asignando memoria a los distintos procesos de dicha cola que se irán cargando en ese espacio y comenzaran a competir por el CPU hasta que ya no haya ningún bloque de memoria disponible (o agujero). En este caso el sistema operativo puede esperar a que se libere uno o puede examinar el resto de la cola

de entrada para ver si puede satisfacer los requisitos de memoria de algún otro proceso que necesite un bloque de memoria menor. Cuando es necesario asignarle a un proceso un agujero, el sistema operativo busca entre los agujeros disponibles para encontrar uno suficientemente grande para albergar al proceso. Si el agujero es demasiado grande se lo dividirá en dos partes, un para el proceso y otra para el conjunto de agujeros, cuando el proceso termine devolverá su bloque de memoria que volverá a colocarse en el conjunto de agujeros, si este nuevo agujero es adyacente a otros se combinaran para formar agujeros más grandes, luego el sistema operativo deberá corroborar si este nuevo agujero puede ser utilizado por algún otro proceso de la cola de entrada.

Asignación de agujeros en general:
Este tipo de mecanismos tienen que ver con como satisfacer una solicitud de tamaño "n" a partir de una lista de agujeros libres. Nombraremos a continuación los tres esquemas de asignación de agujeros más utilizados:

- ✓ Primer ajuste: se asigna el primer agujero que sea lo suficientemente grande. La exploración puede comenzar siempre desde el principio, o desde donde dejo la ultima exploración
- ✓ Mejor ajuste: se asigna el agujero más pequeño que tenga el tamaño suficiente, se explora la lista completa, a menos que este ordenada según su tamaños.
- ✓ Peor ajuste: se asigna el agujero más grande que tenga el tamaño suficiente, se explora la lista completa, a menos que este ordenada según tamaños.

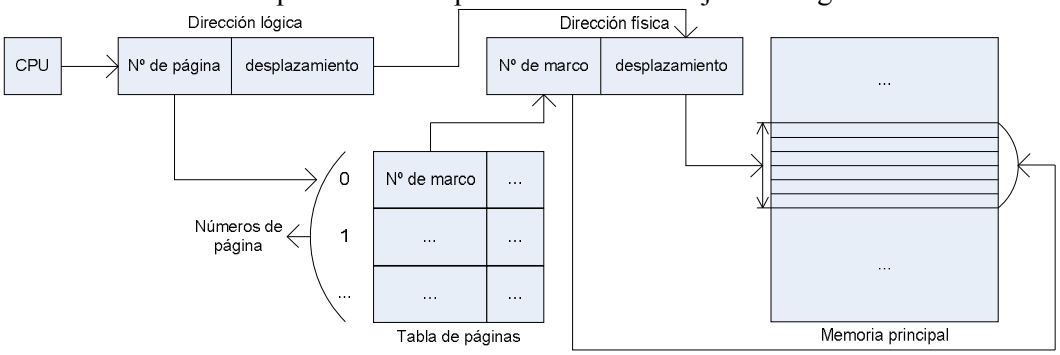
Las simulaciones encuentran que la primera y la segunda son las mejores técnicas en términos de tiempo de asignación y utilización del espacio de almacenamiento, no estando claro cuál de estas dos es mejor en términos de utilización del espacio, pero se sabe que siempre la primera de las dos es la más rápida de implementar.

Fragmentación:
Tanto en el esquema del primer ajuste como en el del mejor ajuste está presente el fenómeno de la fragmentación externa e interna. La estrategia del primer ajuste es mejor en cuanto a este fenómeno para algunos tipos de sistemas, y la del mejor ajuste para otros. También influye en este fenómeno cual es el extremo que se dejara como agujero libre del agujero que se asignara, si la parte superior o inferior del mismo. Una posible solución a la fragmentación externa es la compactación, la cual consiste en movilizar toda la memoria ocupada para que toda la memoria libre quede en un solo espacio contiguo, sin embargo esta alternativa es solo aplicable en los esquemas de carga dinámicos Pero incluso en su expresión más simple esta solución tiene un alto coste en cuanto al rendimiento del sistema. Otra posible solución al problema de la fragmentación externa, consiste en permitir que el espacio de direcciones lógicas de los procesos no sea contiguo. Existen dos esquemas que permiten implementar este tipo de asignación, que veremos a continuación; la paginación y la segmentación, además de la combinación de ambas.

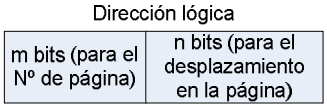
Paginación:
Es un esquema de asignación no contiguo, como dijimos antes y evita los problemas de los esquemas anteriores de fragmentación externa, y encajar fragmentos de tamaño variable en el almacén de respaldo, generando todos los problemas antes vistos en la memoria, pero en el almacenamiento de respaldo, donde por ejemplo, la compactación

es imposible por la velocidad mucho menor de acceso. En la mayoría de los casos la paginación se gestionaba mediante hardware, sin embargo algunos diseños recientes implementan paginación integrando estrechamente hardware y software del sistema operativo, especialmente en arquitecturas de 64 bits.

Modelo básico:
El esquema básico implica dividir la memoria física en un conjunto de bloques de tamaño fijo denominados marcos, y la memoria lógica en otro conjunto de bloques fijos del mismo tamaño denominados paginas. El almacén de respaldo se divide en un conjunto de bloques del mismo tamaño que los marcos de la memoria, y cuando hay que ejecutar un proceso sus páginas se cargan desde el almacena de respaldo en los marcos de memoria disponibles. El soporte hardware trabaja de la siguiente manera:



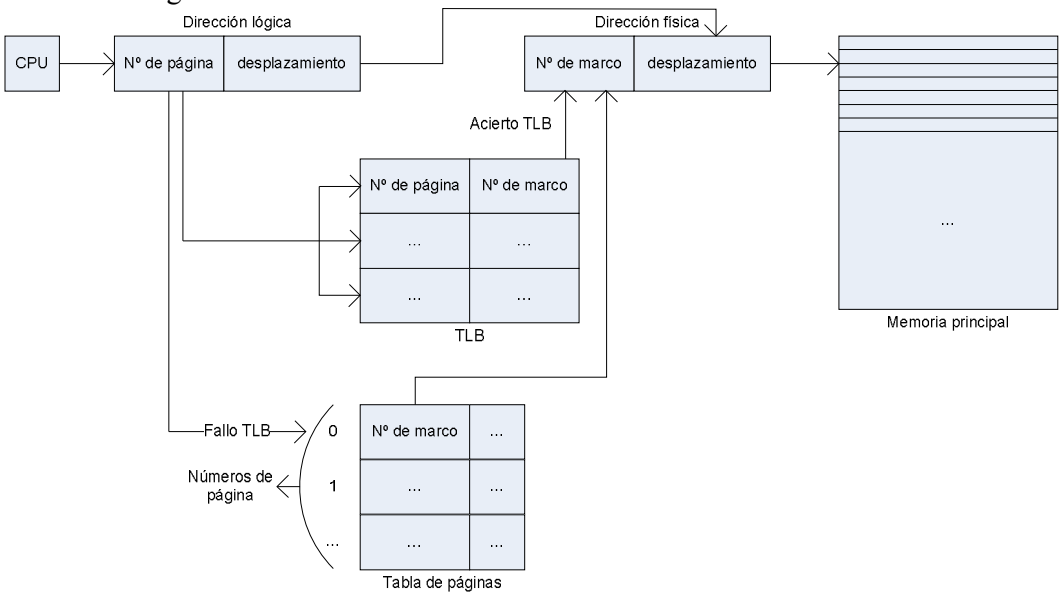
Como se ve en la figura el numero de pagina es el índice para acceder a la "Tabla de paginas" que almacena las direcciones base de los marcos con los que se corresponden las paginas, estas direcciones base se combinan con los desplazamientos de pagina para definir la dirección de memoria física que se envía a la unidad de memoria. El tamaño de página lo define el hardware y es del rango comprendido entre 512 B y 16 MB, pero siempre en potencias de "2", esto último para lograr direcciones lógicas con el siguiente formato:



A pesar de que este esquema no posee fragmentación externa, si posee interna, en los casos en los que los requisitos de memoria de un proceso no sean de tamaño múltiplo del tamaño de la pagina, habrá fragmentación interna en la última página. Se debe hacer un balance entre la fragmentación interna, los recursos utilizados para gestionar mayores cantidades de páginas y la mejoría en la eficiencia en las operaciones de E/S entre disco y memoria, para tamaños más voluminosos de datos, ese balance debe converger en un tamaño adecuado de pagina. Sin embargo, con el tiempo, el tamaño de página ha ido aumentando. Cuando un proceso es seleccionado para ejecutar, se examina el tamaño de este en páginas, si hay tantos marcos disponibles como paginas necesita el proceso, se le asignaran estos al proceso de a uno a la vez, escribiendo en la tabla de páginas del proceso la correspondencia entre cada página y cada marco asignado, de la forma que dijimos anteriormente. Puesto que el sistema operativo es el gestionador de marcos, deberá saber de los marcos, cuestiones como si están asignados, si están disponibles, cual es el número total de marcos, etc. Esta información se encuentra en la denominada "Tabla de marcos", esta contiene una entrada por cada marco e indica por cada una de ellas si el marco está asignado o libre y de estar asignado, a que pagina de que proceso o procesos ha sido asignada El sistema operativo posee una copia de la "Tabla de paginas" de cada proceso, y la utiliza para traducciones

manuales de direcciones lógicas a direcciones físicas y para el despachador que utiliza dicha copia para definir la “Tabla de paginas hardware” en el momento de asignar la CPU a un proceso.

Soporte hardware:
Cada sistema operativo tiene sus propios métodos para almacenar tablas de páginas, la mayoría de ellos tienen una tabla por proceso almacenando, un puntero a la misma junto con los otros valores de los registros en el PCB. El despachador carga de la PCB los registros en la CPU, y arma la “Tabla de paginas hardware” a partir de la tabla almacenada de paginas cuyo puntero esta también en la PCB, todo esto para iniciar un proceso. En el hardware la tabla de páginas, en su exponente de implementación más sencillo, será un conjunto de registros dedicados. Las instrucciones que utiliza el despachador para cargar o modificar los registros de la tabla son por su puesto, privilegiados. Pero este esquema hoy en día no es posible, dada la cantidad de páginas que utiliza cada proceso, es por esto que la alternativa consiste en tener un único registro que contenga la base de la tabla en memoria llamado “Page table base register” (PTBR). Para cambiar la “Tabla de páginas” ahora solo se necesita cambiar el valor del PTBR para que apunte a otra base, reduciendo sustancialmente el cambio de contexto, que antes implicaba llenar todos los registros dedicados. El problema radica en el tiempo requerido para acceder a una ubicación de memoria del usuario, con este esquema se harían dos accesos a memoria por cada byte, uno con el PTBR y el desplazamiento para acceder a la entrada de pagina necesaria y obtener su marco, y otra para con el marco y el desplazamiento acceder al byte, lo cual es intolerablemente ineficiente. La solución consiste en utilizar una cache de hardware especial, de pequeño tamaño y con capacidad de acceso rápido llamada “Translation lookaside buffer” (TLB) en la cual cada entrada está compuesta por dos partes; una clave (o etiqueta) y un valor, en la búsqueda se compara el valor de una clave con todas las claves de la tabla y si hay acierto, se devuelve un valor. Esta búsqueda se realiza en paralelo de forma muy rápida, pero la cantidad de entradas de la TLB es escueta (entre 64 y 1024 entradas). La TLB se utiliza de la siguiente manera:



Todo el uso de la TLB ralentiza únicamente 10% más en caso de fallo que si no se hubiese utilizado. Si se produce un fallo se puede cargar una nueva entrada en la TLB para que albergue la entrada que lo produjo por no haberla podido encontrar, de manera

de acceder más rápidamente la próxima vez que se busque. Si la TLB está llena el sistema operativo deberá eliminar una entrada para colocar la nueva, esta sustitución se realiza mediante toda clase de algoritmos. Algunas TLB permiten proteger ciertas entradas para que estas no puedan ser eliminadas, generalmente allí se albergan las páginas de código del kernel. Otro tipo de extras que puede tener una TLB son los "Address space identifier" (ASID), en cada entrada de la TLB. Lo que permite es mejorar por un lado la protección ya que la TLB corrobora en la búsqueda que el ASID de la entrada encontrada sea el mismo que el del proceso que está ejecutando actualmente y no el de otro proceso (dado que cada ASID se asocia a un proceso diferente) y por el otro, permite mantener entradas simultáneamente para varios procesos distintos, que si la TLB no soportara ASID se tendría que vaciar entera en cada cambio de tabla de páginas para que no se produzcan accesos erróneos

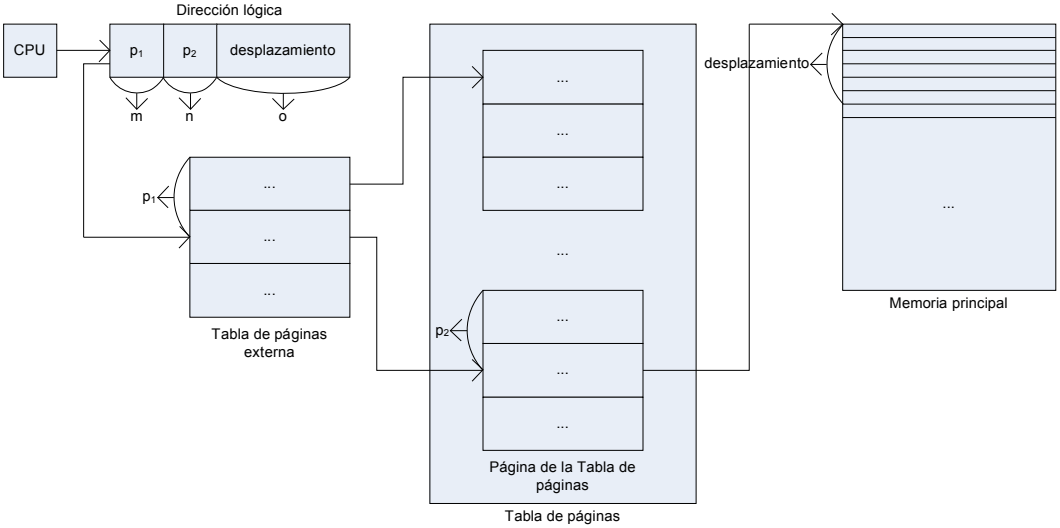
Protección:
Se consigue mediante una serie de bits de protección asociados con cada marco, estos normalmente se mantienen en la tabla de páginas. Podríamos proveer un hardware que proporcionase protección de solo lectura, de lectura y escritura o de solo ejecución, o podemos permitir cualquier combinación de ellos con un bit por cada tipo de acceso, los intentos ilegales provocaran una interrupción de hardware hacia el sistema operativo generalmente lo que pasa es que se asocia un bit adicional con cada entrada en la tabla de páginas, el bit se denomina "Valido-Invalido". Si se pone el bit en valido significara que dicha pagina se encuentra en el espacio de direcciones lógicas del proceso y será por lo tanto una página legal (o valida), si de lo contrario el bit se pone en invalido, la pagina no se encontrara en el espacio de direcciones lógicas del proceso. Algunos sistemas proporcionan mecanismos de hardware especiales en forma de un registro llamado "Page table length register" (PTLR) para indicar el tamaño de la tabla de páginas, para casos en los que no valga la pena que un proceso acceda a todo el rango de direcciones que pueda porque no las utiliza. Este registro se compara con las solicitudes provenientes del CPU, para controlar que el proceso no se salga de su rango, si esto se produce también lo hará una interrupción de error dirigida al sistema operativo.

Paginas compartidas:
Una ventaja de la paginación es la posibilidad de compartir código común, este se denomina "Código reentrante" o puro, y tiene la característica de que no se auto modifica es decir nunca cambia durante la ejecución Esta propiedad de solo lectura del "Código reentrante", debe de ser impuesta por el sistema operativo. Algunos sistemas implementan el mecanismo de comunicación entre procesos llamado "Memoria compartida", de forma similar a la del "Código reentrante".

Estructura de tabla de páginas:
En esta sección veremos algunas de las técnicas más comunes para estructurar la tabla de páginas.

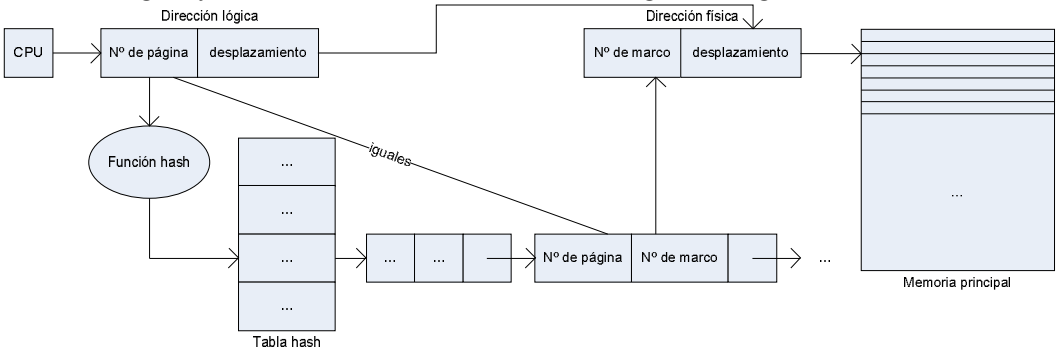
Paginación jerárquica:
Una forma de solucionar la cuestión del excesivo tamaño de la tabla de páginas en los sistemas informáticos modernos, consiste en dividir dicha tabla en fragmentos más pequeños, y la forma de realizarlo, tiene varias alternativas. Una de estas consiste en paginar la propia tabla en un algoritmo de dos niveles o "Tabla de páginas con

correspondencia (mapeo) directa". Para mostrar cómo se implementaría mostramos como quedaría la dirección lógica y como se haría la traducción en la siguiente figura:



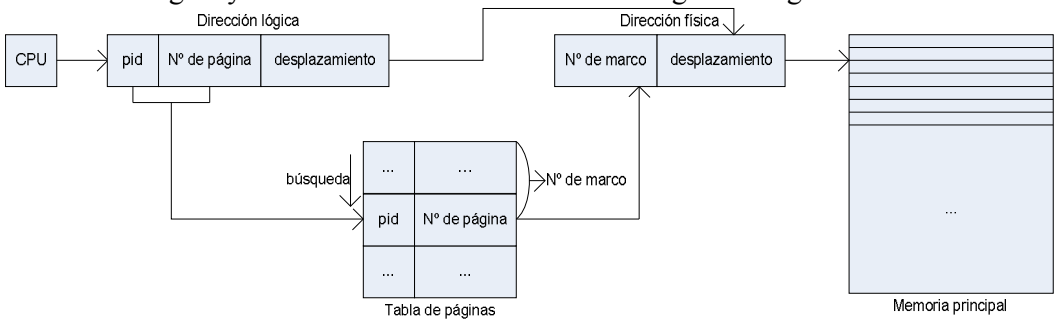
En la figura "m" denota la cantidad de bits necesarios para referenciar una entrada en la tabla de paginas externas, de dicha entrada se obtiene la base de una página en la tabla de páginas, y luego con los "n" bits siguientes de la dirección lógica se realiza un desplazamiento en dicha pagina para acceder al marco de memoria que le corresponde, finalmente con los "o" bits últimos de la dirección lógica se accede a la posición dentro del marco en memoria y por lo tanto al byte deseado. La cantidad de niveles de este esquema se puede aumentar hasta "n", pero en general para arquitecturas de 64 bits las tablas de páginas jerárquicas se consideran inapropiadas.

Tabla de páginas hash:
Una técnica común para gestionar espacios de direcciones superiores a 32 bits consiste en utilizar una "Tabla hash de paginas", en la cual cada entrada contiene una lista enlazada de elementos que tienen un valor hash en común, cada elemento de la lista tiene un numero de pagina, el valor de su correspondiente marco en memoria y un puntero al siguiente elemento de la lista. Para mostrar cómo se implementa indicamos la dirección lógica y como se haría la traducción en la siguiente figura:



Como se ve en la figura al número de página de la dirección lógica se le aplica una función hash, que da por resultado la dirección de una entrada de la tabla hash, una vez que se accede a dicha entrada, se recorre la lista enlazada a la que apunta, comparando el numero de pagina de cada elemento con el de la dirección lógica, cuando hay acierto, se utiliza el marco de memoria almacenado en ese elemento y el desplazamiento de la dirección lógica para construir la dirección física y acceder al byte en memoria.

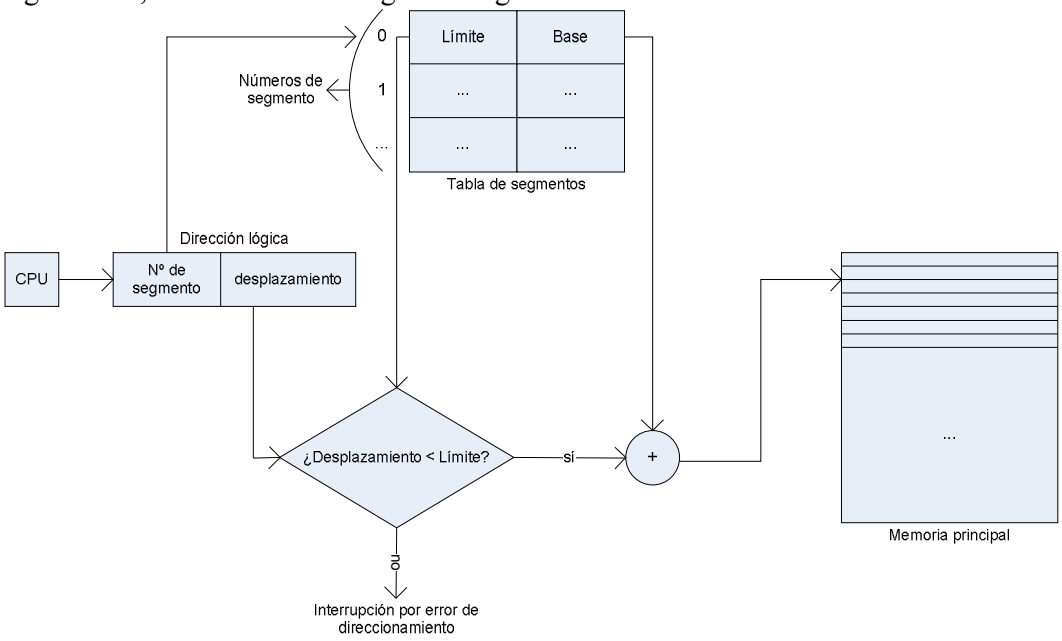
Tabla de páginas invertida:
Usualmente cada proceso tiene una tabla de paginas asociada, dicha tabla de paginas incluye una entrada por cada página del proceso. Esta representación resulta natural, sin embargo las tablas pueden ocupar una gran cantidad de memoria física, es por esto que se utiliza una tabla de páginas invertida. Para mostrar cómo se implementa indicaremos la dirección lógica y como se haría la traducción en la siguiente figura:



Como se ve en la tabla hay una entrada por cada marco de memoria, en cada una de ellas hay información del proceso al que se le asigno dicho marco y con qué pagina dentro del proceso se corresponde. Para realizar la traducción se explora la tabla de páginas buscando una correspondencia entre los dos primeros campos de la dirección lógica, y la entrada, si se encuentra en una posición determinada, el número de esa posición más el desplazamiento de la dirección lógica, serán la dirección física, de no encontrarse correspondencia se tratara de un acceso ilegal. Como variante de este esquema se puede utilizar una tabla hash como la que describimos con anterioridad, con el fin de limitar la búsqueda a una sola entrada en la tabla de páginas o como mucho a unas pocas entradas. Este esquema tiene dificultades para implementar el concepto de memoria compartida, dado que bajo este esquema un mismo marco no puede tener más de una página, como en general así se implementa la memoria compartida, de ahí su dificultad. Una alternativa seria permitir que la tabla de paginas solo contenga una única correspondencia de una dirección virtual con la dirección física compartida, esto implicaría que las referencias a direcciones virtuales que no estén asociadas darán como resultado fallo de pagina.

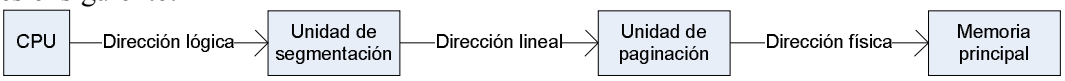
Segmentación:
Es un esquema de gestión de memoria que soporta la visión de la memoria de los usuarios, en este esquema un espacio lógico de direcciones es una colección de segmentos, cada uno de ellos con nombre y longitud. Las direcciones especifican tanto el nombre del segmento como el desplazamiento dentro del mismo. Decimos nombre para dar la idea de identificador, en realidad cada segmento se denota mediante un número de segmento. Normalmente es el compilador el encargado de construir automáticamente los segmentos para reflejar el programa de entrada.

Hardware:
Para mapear las direcciones lógicas en las físicas se utilizan una denominada “Tabla de segmentos”, como muestra la siguiente figura:

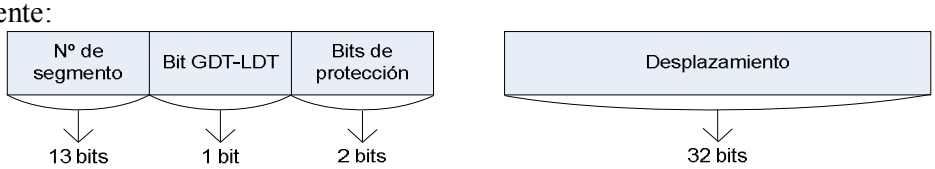


Como se puede ver en la figura mediante el numero de segmento como índice se accede a la entrada correcta de la tabla de segmentos, luego se compara el valor del límite del segmento alojado en dicha entrada con el valor del desplazamiento de la dirección física, para corroborar que no hay un acceso ilegal fuera del segmento, si lo hay se produce una interrupción hacia el sistema operativo, si no lo hay mediante la base de la entrada contenido en la tabla, se fija en la memoria la primera posición del segmento, para llegar al byte deseado se le suma al valor de la base el del desplazamiento y se accede.

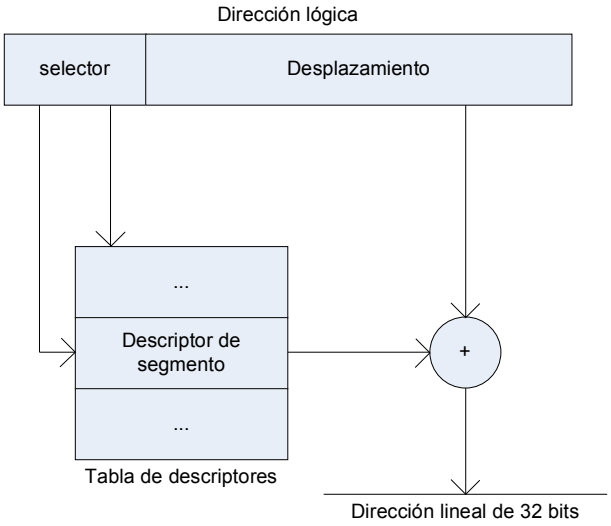
Ejemplo Intel Pentium:
Esta arquitectura soporta ambos mecanismos, el de segmentación y el de paginación, como así también el de segmentación paginada. El esquema de traducción en el Pentium es el siguiente:



Segmentación en Pentium:
Este CPU permite que un segmento tenga un tamaño de hasta 4 GB y permite una cantidad de segmentos por proceso de 16 KB de esa cantidad, la mitad son segmentos privados del proceso, y la otra mitad son compartidos por todos los procesos, la información de la primera partición se almacena en la “Local descriptor table” (LDT) y la de la segunda en la “Global descriptor table” (GDT), cada entrada de estas tablas está compuesta por un descriptor de segmento de 8 B que incluye información detallada del segmento concreto, incluyendo la base y limite del mismo. La dirección lógica es la siguiente:

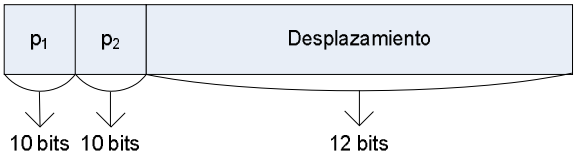


La CPU tiene seis registros de segmento lo que permite al mismo direccionar en cada momento hasta seis segmentos y también seis registros de 8 B cada uno para almacenar los correspondientes descriptores LDT o GDT. El mecanismo de traducción se muestra en la siguiente figura:

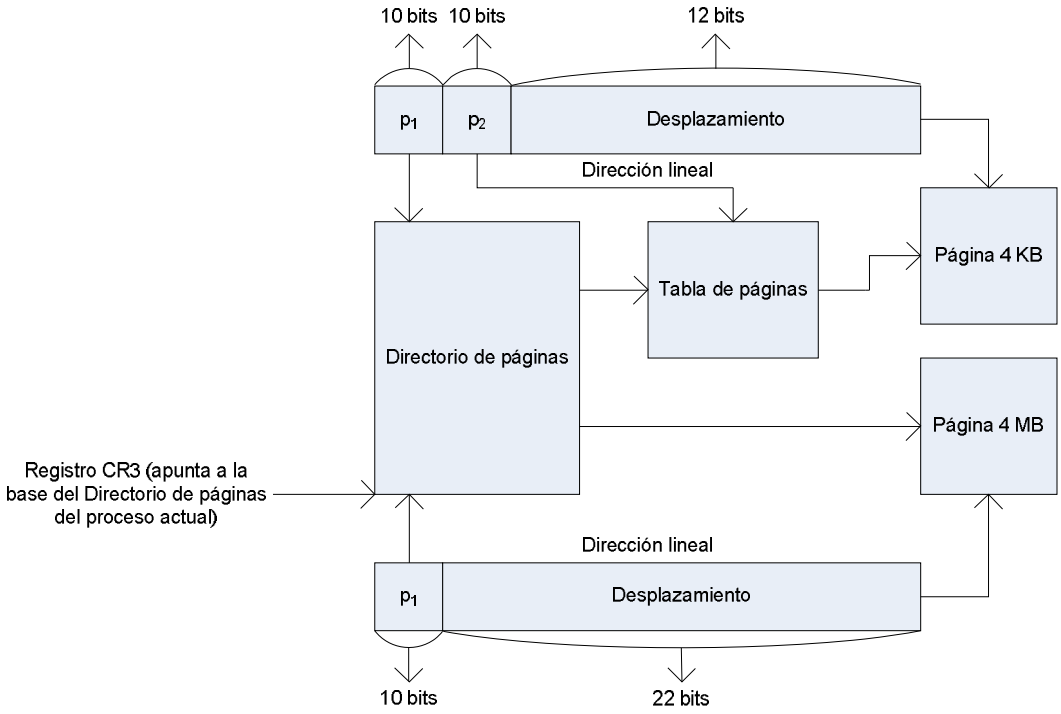


El registro de segmento apunta a la entrada apropiada de la LDT o de la GDT, la información de limite a cerca del segmento se compara con el desplazamiento, si este es más grande que el mismo, se genera una interrupción hacia el sistema operativo, y si es menor entonces se le suma a la base el desplazamiento, y se obtiene una dirección lineal de 32 bits que será procesada por la unidad de paginación

Mecanismo de paginación:
El Pentium permite páginas de 4 KB o 4 MB, para las de 4 KB e CPU utiliza un esquema de dos niveles en el que la estructura de la dirección lineal de 32 bits es la siguiente:



La tabla de paginas externa se denomina aquí “Directorio de paginas”. El mecanismo de traducción es idéntico al mecanismo mencionado en la sección de “Paginación jerárquica” por esta razón la omitiremos y solo mostraremos la figura de la que se puede deducir:



Capitulo 9:
Memoria Virtual

La técnica de memoria virtual es un mecanismo que permite la ejecución de procesos que no se encuentran completamente en la memoria.

Fundamentos:
Existe un requisito fundamental de la ejecución de un proceso, y es que sus instrucciones deben estar en la memoria para poder ejecutarse, lo cual limita al proceso a no exceder el tamaño de la memoria física. Sin embargo, un examen de los programas reales nos muestra que en muchos casos no es necesario tener el programa completo para ejecutarlo, por diversas razones, o en casos donde si se necesite tenerlo completo puede que no todo el programa al mismo tiempo. Las ventajas de poder ejecutar un programa que solo se encuentre parcialmente en memoria son:

- ✓ Los programas ya no estarán restringidos por la cantidad de memoria física disponible.
- ✓ Se podrán ejecutar más programas al mismo tiempo.
- ✓ Se necesitaran menos operaciones de E/S para cargar o intercambiar cada programa de usuario.

La memoria virtual incluye la separación de la memoria lógica tal y como la percibe el usuario, de la memoria física. Esto le permite al mismo despreocuparse de la cantidad de memoria física disponible y en lugar de ello concentrase en el problema que debe resolver mediante su programa. También permite que dos o más procesos compartan archivos y memoria mediante mecanismos de compartición de páginas, esto proporciona las siguientes ventajas:

- ✓ Las bibliotecas del sistema pueden ser compartidas por numerosos procesos, mapeando el objeto compartido sobre un espacio de direcciones virtuales, normalmente en modo de solo lectura.
- ✓ Permite implementar un mecanismo de comunicación entre procesos por memoria compartida.
- ✓ Permite que se compartan páginas durante la creación de un proceso mediante llamadas como Fork( ); .

Paginación bajo Demanda:
Este mecanismo de carga de un programa del disco a la memoria consiste en cargar las páginas de dicho programa, únicamente cuando sean necesarias, durante la ejecución del programa. De esta forma nunca se cargaran aquellas páginas a las que nunca se acceda. Este mecanismo es similar al de paginación con intercambio, solo que en lugar de intercambiar procesos completos, lo que hace es un “intercambio perezoso”, lo que implica como antes dijimos que jamás se intercambiara una página con memoria a menos que esta vaya a ser necesaria.

Conceptos Básicos:
Cuando hay que cargar un proceso, el paginador realiza una estimación de que páginas serán utilizadas antes de descargar de nuevo el proceso. Con este esquema necesitamos

un soporte hardware para distinguir entre las páginas que se encuentran en memoria y las páginas que residen en el disco. Se puede utilizar el antes mencionado bit de "valido-invalido", donde redefiniendo el bit, valido significaría página legal e invalido significaría página no legal o bien página valida pero no presente en memoria, por lo tanto según este esquema en cada entrada de la tabla de páginas se tendrá en caso de no estar en la página presente en memoria el bit de invalido activado o se tendrá la dirección de la página dentro del disco. Mientras el proceso ejecute y acceda a las páginas que estén residentes en memoria (es decir, según lo dicho anteriormente, las más necesarias, que tienen el bit de validas activados), la ejecución se llevara a cabo normalmente, pero cuando se intente acceder a una página con el bit de invalido activado, se provocara una interrupción de fallo de página (o como antes dijimos el hardware realizara una interrupción dirigida al sistema operativo al detectar el bit de invalido como activado), la cual podrá ser tratada de la siguiente manera :

- ✓ Comprobamos en una tabla interna (que usualmente se mantiene con el PCB) correspondiente a este proceso para determinar si la referencia era un acceso de memoria valido o invalido.
- ✓ Si era invalido, terminamos el proceso. Si era valido pero todavía la página no ha sido cargada en memoria, la cargamos.
- ✓ Buscamos un marco libre.
- ✓ Ordenamos una operación de disco para leer la página deseada en el marco recién asignado.
- ✓ Una vez completada la lectura de disco, modificamos la tabla interna que se mantiene con los datos del proceso y la tabla de páginas para indicar que dicha página se encuentra ahora en memoria.
- ✓ Reanudamos la instrucción que fue interrumpida, el proceso podrá ahora acceder a la página, como si siempre hubiese estado en memoria.

En un caso extremo podríamos empezar a ejecutar un proceso sin ninguna página en memoria, este esquema seria una paginación bajo demanda pura; nunca cargar una página en memoria hasta que sea requerida. Sin embargo, la degradación del sistema seria poco tolerable, afortunadamente el análisis de la ejecución de los procesos muestra que este comportamiento consistente en, por ejemplo, acceder a varias nuevas páginas de memoria con cada ejecución de instrucción, es poco probable que se manifieste, con lo que las prestaciones que pueden obtenerse con el mecanismo de paginación bajo demanda son razonables. El hardware necesario para implementar este mecanismo es el mismo que para paginación e intercambio:

- ✓ Una tabla de páginas.
- ✓ Memoria secundaria.

Otro requisito fundamental es el de poder reiniciar cualquier instrucción después de un fallo de página. Si el fallo de página se produce al extraer una instrucción, podemos reanudar la operación volviendo a extraer dicha instrucción, si se produce mientras estamos leyendo una operando, se deberá extraer y decodificar de nuevo la instrucción y luego volver a leer el operando.

Rendimiento de la Paginación bajo demanda:
La paginación bajo demanda puede afectar significativamente el rendimiento de un sistema informático. Para mostrar de qué forma lo hace, utilizaremos la siguiente expresión:

$$t_{ae} = (1-p)\, A_m + p\, t_{fp}$$

Donde:

t_{ae}: es el tiempo acceso efectivo a una memoria con paginación bajo demanda.

A_m: es el tiempo de acceso a memoria (10 ns a 200 ns).

$0<p<1$: es la probabilidad que se produzca fallo de página, cabria esperar que fuera próxima a "0".

t_{fp} : es el tiempo de fallo de página.

En cualquier caso nos enfrentamos con tres componentes principales que influyen en nuestra estimación de la variable "t_{fp}":

- Servir la interrupción de fallo de página.
- Leer la página.
- Reiniciar el proceso.

Las tareas primera y última, pueden reducirse codificando cuidadosamente el software, a unos pocos cientos de instrucciones, y pueden tomar entre 1 y 100 microsegundos cada uno, sin embargo la segunda tarea será del orden de los 8 milisegundos, por lo tanto, despreciando los tiempos de las tareas primera y tercera, el valor de "t_{fp}", será próximo a los 8 milisegundos, si además tomando el peor caso para "A_m" es decir 200 ms, la expresión anterior será:

$$t_{ae} = ((1-p)\, 200 + p\, 8000000)\ ms$$

$$t_{ae} = (200 + 7999800\, p)\ ms$$

Si deseamos que la degradación del rendimiento sea inferior al 10% entonces:

$$220\ ms > (200 + 7999800\, p)\ ms \Rightarrow p < 0{,}0000025000625$$

El resultado implica que por cada 399990 accesos a memoria, uno solo puede ser un fallo de página. En resumen resulta crucial mantener una tasa de fallos de página baja en los sistemas de paginación bajo demanda. Otro aspecto adicional en cuanto al rendimiento es la gestión y el uso global del espacio de intercambio, cuyas operaciones de E/S a disco son generalmente más rápidas que las correspondientes al sistema de archivos, el sistema puede entonces obtener una mayor tasa de transferencia para paginación copiando la imagen completa de un archivo en el espacio de intercambio, en el momento de iniciar el proceso, y luego realizando una paginación bajo demanda, a partir del espacio de intercambio. Otra opción consiste en demandar las páginas inicialmente al sistema de archivos, pero irlas escribiendo en el espacio de intercambio a medida que se las sustituye. Esta técnica garantiza que solo se leen desde el sistema de

archivos las páginas necesarias y que todas las operaciones de paginación subsiguientes se llevaran a cabo desde el espacio de intercambio. Otros sistemas intentan limitar la cantidad de espacio de intercambio utilizado, mediante mecanismos de paginación bajo demanda de archivos binarios, las páginas demandadas de tales archivos se cargan directamente del sistema de archivos. Sin embargo, cuando hace falta sustituir páginas, estos marcos pueden simplemente sobrescribirse (dado que nunca se han modificado), y las páginas pueden volver a leerse desde el sistema de archivos en caso necesario. Sin embargo seguirán habiendo páginas en dicho espacio como las de pila y cúmulo de cada proceso (que si se pueden modificar).

Copia durante la escritura:
Habíamos dicho antes que un proceso podía comenzar rápidamente cargando únicamente en memoria la página que contuviera su primera instrucción a ejecutar, sin embargo mediante Fork(); y utilizando una técnica similar a la de compartición de páginas, se puede minimizar aun más el número de nuevas páginas que se deben asignar al proceso y con ello acelerar también la creación del mismo. Durante esta llamada, es posible utilizar una técnica conocida con el nombre de "copia durante la escritura" que permite que padre e hijo compartan inicialmente las mismas páginas, estas se marcan como "páginas de copia" durante la escritura, lo que implica que si cualquiera de los procesos escribe en una de las páginas compartidas se creara una copia de esa página compartida. Otra alternativa a la llamada Fork (); es la llamada Vfork(); (o fork para memoria virtual), con ella no se utiliza un mecanismo de copia durante la escritura, si no que el proceso hijo utiliza el espacio de direcciones del padre, y este último se suspende. Si el hijo realiza modificaciones, el padre podrá visualizarlas una vez que reanude su ejecución, es por esto que Vfork(); es muy eficiente para la creación de procesos, y es por esto que generalmente se lo utiliza cuando los procesos hijos ejecutaran Exec(); inmediatamente después de su creación.

Sustitución de páginas:
La sobre asignación de memoria se manifiesta, en los casos en los que se está ejecutando un proceso y se produce un fallo de página, pero no hay ningún marco libre en la lista de marcos libres, es decir toda la memoria está siendo utilizada. En casos como este el sistema operativo podrá:

- ✓ Terminar el proceso de usuario que produjo el fallo de página, lo cual es lo que precisamente intenta mejorar la paginación bajo demanda, entonces no será generalmente la mejor alternativa.
- ✓ Descargar un proceso de memoria liberando todos los marcos que le correspondan y reducir así el nivel de multiprogramación. que es una técnica que analizaremos más adelante.
- ✓ El mecanismo de sustitución de páginas, que describiremos a continuación.

Sustitución Básica de páginas:
La idea de este mecanismo consiste en que si no hay ningún marco de memoria disponible, se localiza uno que no esté siendo actualmente utilizado y se lo libera. Para liberar un marco podemos escribir su contenido en el espacio de intercambio y modificar la tabla de páginas y las demás tablas para indicar que esa página ya no se encuentra en memoria. Modificamos la rutina de servicio de fallo de página para incluir el mecanismo de sustitución de páginas:

- ✓ Hallar la ubicación de la página en disco.
- ✓ Localizar un marco libre :
    - Si hay uno utilizarlo
    - Si no hay uno, utilizar un algoritmo de sustitución de páginas para seleccionar un marco víctima.
    - Escribir el marco victima en el disco y cambiar las tablas de páginas y marcos correspondientemente.
- ✓ Leer la página deseada y cargarla en el marco liberado y cambiar las tablas de páginas y marcos.
- ✓ Reiniciar el proceso de usuario.

Para no realizar siempre dos transferencias de página (carga y descarga), si no hay un marco libre, podemos añadir un bit de modificado o sucio, en el hardware de cada marco para que si no se modifica la página (es decir si no tiene dicho activo), o si esta es de solo lectura no sea necesario actualizarla en disco. Utilizando este bit solo se realizaría una única transferencia (la de carga), sin embargo, si la página fue modificada (es decir si tiene el bit activado), entonces no habrá otra opción más que realizar las dos transferencias.

Existen entonces dos problemas importantes para la paginación bajo demanda:

- ✓ Un algoritmo de asignación de marcos.
- ✓ Un algoritmo de sustitución de páginas.

El criterio más importante para elegir que algoritmo implementar, en el caso del segundo problema consiste en determinar cuál es algoritmo que produzca menos fallos de página. Podemos evaluar la cantidad de fallos de página producidos por un algoritmo aplicándolo a una cadena determinada de referencias de memoria. La forma de generar cadenas de memoria es:

- ✓ Artificialmente.
- ✓ Mediante una traza en un sistema determinado.

Para reducir los datos de las referencias a memoria podemos utilizar dos hechos:

- ✓ Solo se considera de ellas el número de páginas.
- ✓ Una vez referenciada una página cuyo número sea “p” en la cadena de referencias, todas las direcciones subsecuentes a esa, que referencien el mismo número de página, hasta que aparezca una cuyo número de página sea distinto, se omitirán como si no se hubiesen solicitado, dado que son incapaces de producir un fallo de página.

También para poder evaluar un algoritmo de sustitución de páginas, será necesario conocer los números de marcos disponibles.

Sustitución de páginas por FIFO:
Es el algoritmo más simple de implementar, la idea FIFO de “la primera página que entra es la primera página que sale”, tiene que ver con el momento en que se carga en memoria la página. A modo de ejemplo evaluamos la cantidad de fallos de página producidos por este algoritmo sobre una cadena cualquiera de referencia:

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	2	2	2	2	4	4	4	0	0	0	0	0	0	0	7	7	7
	0	0	0	0	3	3	3	2	2	2	2	2	1	1	1	1	1	0	0
		1	1	1	1	0	0	0	3	3	3	3	3	2	2	2	2	2	1
X	X	X	X		X	X	X	X	X	X			X	X			X	X	X

Total de Fallos de Página = 15

El rendimiento de este algoritmo no siempre es bueno, por ejemplo, después de sustituir una página activa por otra nueva, se producirá casi inmediatamente un nuevo fallo de página que provocar la recarga de la página activa, de este modo una mala elección de la página que hay que sustituir incremente la tasa de fallos de página, y por lo tanto ralentiza la ejecución de los procesos.

Anomalía de Belady: para algunos algoritmos de sustitución de páginas, la tasa de fallos de página puede incrementarse a medida que se incrementa el número de marcos asignados. Para mostrar un ejemplo tomamos para un algoritmo FIFO, la siguiente cadena de referencias en un sistema de 3 marcos y otro de 4 marcos respectivamente:

1	2	3	4	1	2	5	1	2	3	4	5
1	1	1	4	4	4	5	5	5	5	5	5
	2	2	2	1	1	1	1	1	3	3	3
		3	3	3	2	2	2	2	2	4	4
X	X	X	X	X	X	X			X	X	

Total de Fallos de Página = 9

1	2	3	4	1	2	5	1	2	3	4	5
1	1	1	1	1	1	5	5	5	5	4	4
	2	2	2	2	2	2	1	1	1	1	5
		3	3	3	3	3	3	2	2	2	2
			4	4	4	4	4	4	3	3	3
X	X	X	X			X	X	X	X	X	X

Total de Fallos de Página = 10

Sustitución optima de páginas:
Como consecuencia de la Anomalía de Belady, se produjo una búsqueda de lo que se dio a llamar como “algoritmo optimo de sustitución de páginas”, el cual no padece de dicha anomalía, y es el algoritmo de menor tasa de fallos de página de entre todos los algoritmos existentes. Este algoritmo sustituye la página que no vaya a ser utilizada durante el periodo más largo. A modo de ejemplo evaluamos la cantidad de fallos de página producidos por este algoritmo sobre una cadena cualquiera de referencia:

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	2	2	2	2	2	2	2	2	2	2	2	2	2	2	7	7	7
	0	0	0	0	0	0	4	4	4	0	0	0	0	0	0	0	0	0	0
		1	1	1	3	3	3	3	3	3	3	3	1	1	1	1	1	1	1
X	X	X	X		X		X			X			X				X		

Total de Fallos de Página = 9

Sin embargo el algoritmo optimo de sustitución de páginas resulta difícil de implementar porque requiere un conocimiento futuro de la cadena de referencia, entonces se opta por utilizarlo principalmente con propósitos comparativos, es decir para determinar que algoritmo se acerca más a la cantidad de fallos de página que produce (dado que es el algoritmo que menos fallos de página produce).

Sustitución LRU (Least-Recently-Used) de páginas:
Este algoritmo reemplaza la página que no haya sido utilizada durante el periodo más largo de tiempo. Para implementarlo se asocia a cada página el instante correspondiente al último uso de dicha página, en el momento de seleccionar la víctima, LRU elige aquella página cuyo instante de ultimo uso sea el mayor de todos. A modo de ejemplo evaluamos la cantidad de fallos de página producidos por este algoritmo sobre una cadena cualquiera de referencia:

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	2	2	2	2	4	4	4	0	0	0	1	1	1	1	1	1	1
	0	0	0	0	0	0	0	0	3	3	3	3	3	3	0	0	0	0	0
		1	1	1	3	3	3	2	2	2	2	2	2	2	2	2	7	7	7
X	X	X	X		X		X	X	X	X			X		X		X		

Total de Fallos de Página = 12

Este algoritmo es considerado bastante bueno, y se utiliza a menudo, pero necesita una considerable asistencia de hardware, dado que es complicado determinar un orden para los marcos definido por el instante correspondiente al último acceso. Las posibles soluciones son:

- ✓ Contadores: en el caso más simple asociamos con cada entrada en la tabla de páginas un campo de tiempo de uso y añadimos a la CPU un reloj lógico o contador, cada vez que se haga una referencia a memoria se incrementa, y cada vez que se haga una referencia a una página se copia el contendió del reloj en el campo nuevo antes mencionado, que se encuentra en la tabla de páginas. De esta manera podremos sustituir aquella página que tenga en dicho campo el menor valor.
- ✓ Pila: se podría también mantener una pila de números de página, en la cual cada vez que se hace referencia a una de ellas, se extrae la misma de su posición en la pila y se la cola en la primera posición de la misma. De esta forma la página menos recientemente usada será aquella que represente el último elemento de la pila.

Por último cabe aclarar que al igual que el algoritmo optimo, LRU no sufre la Anomalía de Belady.

Sustitución de páginas mediante aproximación LRU:
Pocos sistemas informáticos proporcionan suficiente soporte hardware como para implementar un verdadero algoritmo LRU de sustitución de páginas. Sin embargo algunos de ellos proporcionan algo de ayuda en la forma de un bit de referencia por cada página, este es activado por el hardware cada vez que se hace referencia a una alguna de ellas cada uno de dichos bits está asociado a una entrada en la tabla de páginas. Inicialmente son todos desactivados por el sistema operativo, después de un cierto tiempo, podemos determinar que páginas se han utilizado y cuáles no,

examinando los bits de referencia, pero no podremos saber el orden de utilización de páginas. Esta es la base para muchos algoritmos de sustitución que sean aproximaciones al algoritmo LRU.

Algoritmo de los bits de referencia adicionales:
Podemos obtener información de ordenación adicional registrando los bits de referencia a intervalos regulares, por ejemplo podríamos mantener 8 bits en un byte a intervalos de 100 ms por cada página de una tabla de memoria donde cada uno de dichos bits almacenase el bit de referencia cada 100 ms de derecha a izquierda, generando así en un numero un "historial" de referencias a cada página de los últimos 8 periodos temporales, si tomamos ese byte como un numero entero no signado entonces el más pequeño será el menos recientemente usado. El numero de bits de historial puede variarse y se selecciona para que la actualización sea lo más rápida posible. En un caso extremo este número se reduce a cero dejando el propio bit de referencia, ese algoritmo se denomina de la segunda oportunidad y es el que enunciaremos a continuación.

Algoritmo de segunda oportunidad:
Este algoritmo es básicamente un FIFO, en el cual se asocia a cada página un bit de referencia, y cuando se elige la página para sustituir si dicho bit está en "1" entonces se le da una segunda oportunidad a la página y se le cambia el valor a "0", luego se selecciona la siguiente página FIFO como posible para sustituir, además cada vez que se referencia la página se deberá activar dicho bit, si el mismo está en "0". Una forma de implementar este algoritmo es con una cola circular, teniendo un puntero para indicar cuál es la siguiente página que hay que sustituir, cuando hace falta un marco, el puntero avanza en búsqueda de una página que tenga el bit de referencia en "0", cuando se encuentra, se coloca la nueva página en dicha posición. Este algoritmo se convierte en FIFO, si todos los bits de referencia están activados.

Algoritmo mejorado de segunda oportunidad:
Podemos mejorar el algoritmo de segunda oportunidad considerando el bit de referencia y el bit de modificación de la siguiente manera:

- ✓ (0,0): no se ha utilizado ni modificado la página recientemente (la página que va a sustituirse).
- ✓ (0,1): no se ha utilizado recientemente pero si se ha modificado.
- ✓ (1,0): se ha utilizado recientemente pero está limpia.
- ✓ (1,1): se ha utilizado y modificado recientemente.

Para sustituir una página se utiliza la lógica del algoritmo anterior, pero en lugar de examinar si la página a la que estamos apuntando tiene el bit de referencia en "1", examinamos a que clase pertenece, entonces instalamos la primera página que encontremos en la que clase no vacía más baja.

Sustitución de páginas basadas en contador:
Se mantiene un contador del número de referencias que se hayan hecho a cada página, y esto permite desarrollar los siguientes esquemas:

- ✓ Algoritmo de sustitución LFU (Least-Frequently-Used): en este algoritmo se sustituye la página que tenga el valor más pequeño del contador. Existen problemas con este algoritmo en casos en que la página se utilice con gran

frecuencia durante la fase inicial y luego nunca más, o con mucha menos frecuencia. Una solución consiste en desplazar una posición a la derecha los contenidos del contador a intervalos regulares de tiempo, obteniendo un contador de uso medio con decrecimiento exponencial.

- ✓ Algoritmo de sustitución MFU (Most-Frequently-Used): se basa en que la página de valor de contador más pequeña acaba probablemente de ser cargada en memoria y todavía tiene que ser utilizada, por ende se sustituye aquella cuyo contador sea más alto.

Algoritmo de buffer de páginas:
Además de un algoritmo de sustitución pueden utilizarse en ocasiones, un conjunto compartido de marcos libres, de los cuales se extrae un marco al momento de realizar una sustitución para cargar en la página necesaria. Este mecanismo posibilita que el proceso que produjo el fallo vuelva a arrancar rápidamente, puesto que no es necesario que el marco que realmente produjo el fallo sea realmente descargado, y se cargue en él la página necesaria; sino que, como dijimos, se toma un marco del conjunto compartido de libres. Luego se descarga el marco que si produjo el fallo de página y se lo coloca en el conjunto de marcos libres compartidos. Dos posibles aplicaciones de este mecanismo podrían ser:

- ✓ Un esquema que tenga un conjunto de páginas modificadas al momento en el cual el dispositivo de paginación está inactivo. Se selecciona una página del conjunto y se la escribe en disco, poniendo luego el bit de modificación en "0", de esta forma aumenta la posibilidad de que a la página víctima no sea necesario sustituirla.
- ✓ Un esquema donde se tiene un conjunto de marcos libres al cual se le asocia la página que alojaban anteriormente, puesto que el contenido de un marco no se modifica luego de la escritura del marco en disco, para la sustitución primero se buscara en el conjunto de marcos libres para verificar que la página solicitada no se encuentre ya cargada en alguno de ellos, y si es así se le asignara dicho marco, omitiendo la operación de E/S que hubiese sido necesaria de no aplicar este esquema.

Asignación de marcos:
Intenta responder a la pregunta "Si se tiene una cantidad de "n" marcos libres y "m" procesos ¿Cuántos marcos libres asigno a cada proceso?". En el caso más simple, un sistema monousuario, el sistema operativo acapara un subconjunto de los "n" marcos libres dejando para los procesos "n-l" marcos libres, los cuales en un sistema de paginación bajo demanda pura, se le iría asignando a un proceso en ejecución, por cada fallo de página que este produzca. Para cargar la página numero "(n-l)+1" se deberá elegir una de las "n-l" páginas cargadas en los marcos libres y sustituirla mediante alguno de los algoritmos antes vistos. Cuando el proceso deje de ejecutar devolverá los marcos que haya utilizado durante su ejecución a la lista de marcos libres. También son posibles otras variantes de este esquema pero la estrategia básica esta clara; asegurarle al proceso todos los marcos libres.

Número mínimo de marcos:
Las estrategias de asignación de marcos que planteamos tienen distintas restricciones, como el número máximo de marcos a asignar, que estará restringido por la cantidad total de marcos del sistema, y el número mínimo de marcos, que estará supeditado al

rendimiento, dado que mientras menor sea el valor de dicho numero, mayor será la cantidad de fallos de página, ralentizando la ejecución del proceso, que es lo que analizaremos en esta sección. Llegaremos entonces a la conclusión de que el número mínimo de páginas está definido por la arquitectura informática, dado que debemos tener suficientes marcos como para albergar todas las diferentes páginas a las que una misma instrucción de un proceso pudiera hacer referencia. Esta ultima situación se agrava en los casos en los que la arquitectura le permite al proceso realizar indirecciones de múltiples niveles en las instrucciones, es decir que una instrucción referencia a una dirección indirecta que a su vez referencia a otra referencia indirecta, y así sucesivamente hasta que todas las páginas de la memoria virtual se vean afectadas. Para resolver esta última dificultad se le impone un límite al número de niveles de la indirección, mediante un contador que parte de un numero de indirecciones máximo en el momento de la primera indirección y luego se decrementa cada vez que se realiza una indireccion hasta llegar a "0" donde se produce una interrupción. Esto nos llevaría a concluir que en tales casos el número de marcos mínimos necesarios seria igual al límite de indirecciones máximo decrementado en una unidad.

Algoritmos de asignación:
La primera alternativa que se nos ocurre es la "asignación equitativa", para la cual se le asigna a cada uno de los "m" procesos antes mencionados "n/m" marcos, pudiendo utilizar el resto de dicho cociente como conjunto compartido de marcos libres. El esquema anterior puede producir un desbalance, asignándole a un proceso de pequeño tamaño más marcos de los que necesita, y a uno de gran tamaño menos. Como esquema que resuelve dicha situación, se tiene la "asignación proporcional", la cual tendrá como objetivo asignar a cada proceso la memoria disponible de acuerdo con el tamaño de cada uno. Definimos entonces una expresión que nos permite calcular la cantidad de memoria a asegurarle a un proceso determinado. La expresión es:

$$a_i = V_i \frac{m}{V}$$

Donde:

a_i : es el total de marcos a asignar al proceso "p_i".

V_i: es el total de memoria virtual para el proceso "p_i".

m : es el número total de marcos disponibles.

V: es la cantidad total de memoria virtual, también se puede escribir como "$\sum_{i=1}^{m} V_i$", siendo "m" el número total de procesos.

Obviamente esta expresión se deberá ajustar para que obtengamos un número superior al número mínimo de marcos requeridos por el conjunto de instrucciones especificas de la maquina, sin que exceda el valor "m".
En los dos esquemas antes mencionados, si aumenta el grado de multiprogramación, los procesos alojados originalmente en memoria perderán algunos marcos destinados al nuevo proceso, y viceversa si disminuye el grado de multiprogramación (es decir si termina un proceso). También hemos de recalcar que ninguno de ambos esquemas tiene en cuenta la prioridad del proceso, dado que la misma es importante, se podría utilizar

en el segundo esquema una expresión donde el cociente dependa de la prioridad en vez de del tamaño del proceso, o bien una combinación de ambas.

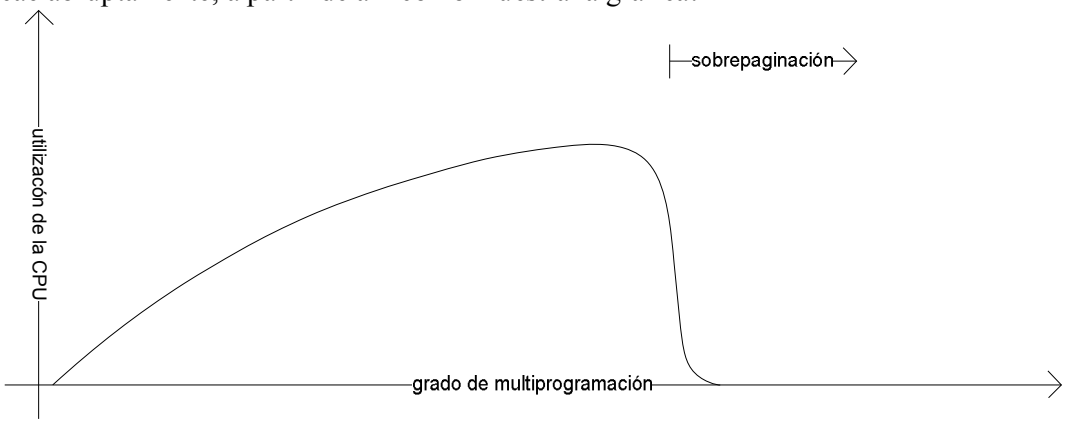
Asignación global y local:
Si hay múltiples procesos compitiendo por los marcos, podemos clasificar los algoritmos de sustitución de páginas en dos categorías amplias:

- ✓ Sustitución global: la cual le permite a un proceso seleccionar un marco de sustitución de entre de el conjunto de todos los marcos, incluso si dicho marco está asignado actualmente a otro proceso.
- ✓ Sustitución local: un proceso solo puede seleccionar un marco de sustitución de entre su propio conjunto de marcos asignados. Por ejemplo, se le podría otorgar la posibilidad de realizar sustitución global a los procesos de alta prioridad. Sin embargo en un esquema global un proceso no puede comprobar su propia tasa de fallos de página, dado que el conjunto de páginas en memoria de un proceso, no solo dependerá de su comportamiento de paginación sino también del comportamiento de paginación de los demás procesos, lo cual no se produce con la sustitución local.

Como desventaja principal la sustitución local perjudicara al proceso, ya que no tendrá a su disposición otras páginas de memoria más que las utilizadas, es por esto que la sustitución global da como resultado una mayor tasa de procesamiento del sistema por lo cual es el método más comúnmente utilizado.

Sobrepaginación:
Un proceso entrara en sobrepaginación, si invierte más tiempo implementando los mecanismos de paginación que en la propia ejecución del mismo. Esto acontece por ejemplo si el numero de marcos asignados a un proceso de baja prioridad cae por debajo del número mínimo requerido por la arquitectura de la maquina, en tal caso debemos suspender su ejecución y descargar de la memoria todas sus páginas. La razón de realizar la suspensión, es que dicho proceso tendrá menos marcos asignados que páginas que utilice de forma activa, entonces cuando se produzca un fallo de pagina porque necesite una de dichas páginas que no esté en memoria, reemplazara otra que también es activa produciendo otro fallo de pagina inmediatamente, y volviendo a realizar el mismo procedimiento, y así sucesivamente sustituyendo páginas que se ve forzado a recargar inmediatamente, lo que producirá graves problemas de rendimiento.

Causa de la sobrepaginación:
A medida que se incrementa el grado de multiprogramación también lo hace la tasa de utilización de la CPU, aunque más lentamente hasta alcanzar un máximo, a partir del cual se produce el fenómeno de la sobrepaginación, y la tasa de utilización de la CPU cae abruptamente, a partir de ahí como muestra la grafica:



En este punto para incrementar la tasa de utilización de la CPU y poner fin a la sobrepaginación es necesario reducir el grado de multiprogramación. Para prevenir la sobrepaginación básicamente debemos proporcionar a los procesos, la cantidad de marcos que necesitan. Para saber cuántos marcos necesita cada proceso existen múltiples alternativas, pero definiremos solo una de ellas, la "localidad de un proceso". La localidad de un proceso es el conjunto de páginas, que utiliza activamente de forma combinada el mismo en un lapso de tiempo, dado que un proceso se va desplazando de una localidad a otra a medida que se ejecuta. Concluimos entonces que las localidades de un programa están definidas por su estructura en sí y por sus estructuras de datos. El modelo de localidad afirma que todos los programas exhibirán esta estructura básica de referencias a memoria. Explicado lo anterior podemos analizar las diferentes formas de evitar la sobrepaginación:

- ✓ Modelo de conjunto de trabajo:
  Este modelo se basa en la creencia de la localidad de ejecución de los programas, antes mencionada. Para aproximar a la localidad de un programa este modelo nos plantea el parámetro "Δ" y el concepto de "conjunto de trabajo". La idea consiste en examinar las "Δ" referencias de página más recientes del proceso y llamar a dichas páginas "conjunto de trabajo". Si una página está siendo utilizada de forma activa, se encontrara en el conjunto, mientras que sino, será eliminada del mismo "Δ" unidades de tiempo después de la última referencia que se hiciera a la misma. La precisión del conjunto de trabajo depende de la selección de "Δ". Si es demasiado pequeña no abarcara la localidad completa, si es demasiado grande puede que se solapen varias localidades. Llamemos TCT_i al tamaño de conjunto de trabajo del i-esimo proceso del sistema, entonces:

$$T = \sum_{i=1}^{m} TCT_i$$

  Donde "T" es la demanda total de marcos, si ¨T¨ se incrementa hasta exceder a ¨m¨
  El sistema operativo seleccionará un proceso y lo suspenderá, descargando sus páginas y repartiendo sus marcos entre otros procesos. Este modelo impide la

sobrepaginación y mantiene el grado de multiprogramación con el mayor valor posible, lo que optimiza la tasa de utilización de la CPU. Para aproximarnos a este modelo podemos utilizar una interrupción de temporización y utilizar un bit de referencia ,copiando y borrando el bit cada vez que se produzca la interrupción en cada página, guardando ocasionalmente un conjunto de bits de referencia más reciente y revisando en dicho historial para un rango de referencia , si se activó alguna vez el bit.

- ✓ Frecuencia de fallos de página:
  Esta estrategia está basada en la frecuencia de fallos de página (PFF, page-fault frequency). Cuando un proceso entra en sobrepaginación se produce un aumento en la PFF, por lo tanto sabemos que el mismo necesita más marcos, y a la inversa, si la PFF disminuye, es decir, puede que el proceso tenga demasiados marcos asignados. La idea de esta estrategia es entonces fijar para la PFF un límite superior a partir del cual se asigna al proceso otro marco y un límite inferior debajo del cual eliminamos un marco del proceso. Al igual que en el método anterior, si no existe un marco disponible y aumenta la PFF deberemos seleccionar un proceso y suspenderlo, otorgando sus marcos liberados a los procesos que tengan mayor PFF.

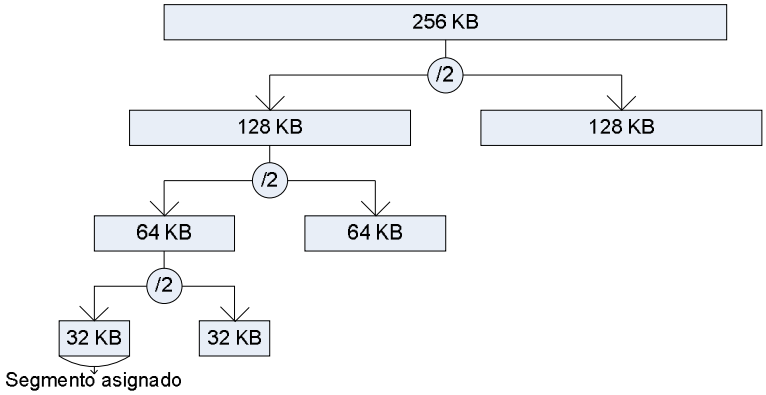
Archivos mapeados en memoria:
Se pueden utilizar las técnicas de memoria virtual explicadas hasta ahora para tratar la E/S de archivo como si fueran accesos rutinarios a memoria .Este recurso es conocido con el nombre de ¨mapeo en memoria¨ de un archivo, y se lleva a cabo mapeando cada bloque de disco sobre una página o varias de la memoria .El acceso inicial se produce provocando un fallo de pagina, entonces se lee una parte del acceso equivalente al tamaño de una página, extrayendo los datos del sistema de archivos
Y depositándolos en la página física (o varias según el sistema operativo del que se trate). De ahí en adelante todas las operaciones de lectura y escritura se gestionarán como accesos normales a memoria. Para actualizar el archivo físico se puede optar por hacerlo inmediatamente, o cuando el sistema operativo compruebe periódicamente si la página fue modificada, de todas formas el cambio en todos los casos se ve plasmado en disco al cerrar el archivo. Algunos sistema operativo incorporan llamada al sistema específicas para tratar con archivos de esta manera, y otros lo realizan independientemente de si se solicita que el archivo sea tratado como mapeado en memoria o no. Mediante este mecanismo obviamente , se puede compartir un archivo por varios procesos ,mapeando concurrentemente el mismo archivo , pudiendo compartirse por lectura y escritura (todos los procesos ven el contenido del archivo y las modificaciones al mismo realizadas por ellos) , o por sólo lectura (todos los procesos pueden ser el contenido pero cada uno dispone de sus propias copias de datos que modifican)Cabe aclarar que en algunos sistema operativo como Windows ,el mecanismo de memoria compartida y el de la compartición de archivos mapeados en memoria ,se implementan bajo el mismo mecanismo.

Asignación de la memoria del Kernel:
La memoria del Kernel suele asignarse a partir de un conjunto compartido de memoria compartida, que es distinto que una lista de marcos libres como la que se utiliza para los procesos. Esto es así por dos razones principales:

- ✓ Las estructuras del Kernel son dinámicas, algunas de ellas tienen tamaño menor a una página, es por esto que el Kernel intenta utilizar la memoria de la forma más conservadora posible y trata de minimizar el desperdicio por fragmentación.
- ✓ Las páginas asignadas al Kernel deben estar en memoria contigua, diferente de los procesos para los que sus páginas pueden estar dispersos en cualquier parte de la memoria.

Estrategias para la gestión de la memoria libre asignada a los procesos del Kernel:

- ✓ Descomposición Binaria (o buddy system):
  Asigna la memoria a partir de un segmento de tamaño fijo, compuesto por páginas físicamente contiguas, mediante una asignación de potencias de ¨2¨ que satisface las solicitudes en unidades cuyo tamaño es potencia de ¨2¨, o buscando el menor de los tamaños mayores que la solicitud que sea múltiplo de ¨2¨. Para mostrar cómo funciona tomamos un segmento fijo de 256 KB, y suponemos que el Kernel solicita 21 KB de memoria, entonces:



  Este esquema tiene como ventaja la rapidez con la que pueden combinarse subsegmentos adyacentes para formar segmentos de mayor tamaño, utilizando una técnica denominada consolidación. En la figura, cuando el Kernel libera el segmento asignado de 32 KB, este se puede ir rearmando hacia arriba con los otros segmentos hasta volver a los 256 KB originales. Sin embargo es un esquema que produce fragmentación interna.
- ✓ Asignación de franjas:
  Una ¨franja¨ está formada por una o más páginas físicamente contiguas, a su vez una ¨caché ¨está formada por una o más franjas .Existe una cache por cada estructura de datos distinta del Kernel, cada una de ellas se rellenan con instancias de la estructura de datos de Kernel que dicha caché representa. Cuando se crea una caché, se le asigna un cierto número de objetos, y estos se marcan inicialmente como libres, este número depende del tamaño de la franja asociada. Cuando hace falta un objeto para una estructura de datos del Kernel, el asignador puede asignar cualquiera de los alojados en la caché que este libre, marcándolo como usado. El asignador de franjas tiene dos ventajas principales:

  - Como dijimos antes cada estructura de datos distinta del Kernel tiene asociada una caché subdividida en franjas, y estas últimas se dividen en segmentos de igual tamaño que las instancias que la estructura de datos del Kernel que representa dicha caché, por lo tanto no se pierde memoria debido a la fragmentación.

- El acto de asignar y liberar memoria para instancias de estructuras de datos del Kernel puede ser un proceso que consume mucho tiempo, sin embargo en este mecanismo con recursos se crean de antemano y pueden por tanto ser asignadas rápidamente a partir de la cache. Además, cuando el Kernel deja de utilizar la instancia la libera y la devuelve a la cache haciendo así que esté inmediatamente disponible para las subsiguientes solicitudes procedentes del mismo. Entonces concluimos que las solicitudes de memoria en un mecanismo como este, pueden satisfacerse muy rápidamente.

Otras consideraciones:
Hay también otras consideraciones además de de la sustitución de páginas y la asignación de marcos, a tener en cuenta en un sistema de paginación:

- ✓ Prepaginación:
  Como ya sabemos, en un sistema de prepaginación bajo demanda, un proceso realizará un gran número de fallos de página cada vez que inicie o retome su cauce de ejecución, y la ¨prepaginación¨ tiene por objetivo evitar esa cantidad de fallos de páginas iníciales. La estrategia consiste en cargar en memoria de una sola vez todas esas páginas que vaya a necesitar un proceso inicialmente .Para decidir si implementar o no la prepaginación se puede tener en cuenta que es más costoso, si atender todos esos Fallos de pagina iníciales o implementar el mecanismo de prepaginación. Un caso donde la implementación resultaría óptima, sería en un sistema que utilice el modelo de conjunto de trabajo, dado que si fuese preciso suspender proceso, recordaríamos el conjunto de trabajo del mismo, y al momento de reanudarlo cargaríamos previamente en memoria el conjunto de trabajo recordado completo para que este no produjese esos fallos de página iníciales.

- ✓ Tamaño de página:
  Los diseñadores de sistemas operativos raramente pueden elegir el tamaño de página, pero cuando se diseñan nuevas plataformas es necesario tomar decisiones en cuanto cual es el tamaño de página más adecuado. Los tamaños de página Son siempre potencias de ¨2¨ y suelen ir generalmente de los 2^{12} B a los 2^{22} B. Una de las cuestiones que influyen en el tamaño de página es el tamaño de la tabla de páginas, dado que si el tamaño de páginas Es muy pequeño, habrá más páginas Y por ende la tabla de páginas Será más grande, puesto que cada proceso activo debe tener una copia propia de la tabla de páginas, conviene utilizar un tamaño de página Grande. Por otro lado, las páginas Se aprovechan mejor si son más pequeñas, dado que es muy probable que un proceso no termine exactamente en una frontera de página, y se produzca fragmentación interna, Concluiremos entonces que para minimizar la fragmentación interna deberemos también minimizar el tamaño de la página. Otra cuestión es el tiempo requerido para leer o escribir una página. Como el tiempo de una de estas operaciones es prácticamente el mismo para páginas de un tamaño determinado, por ejemplo del doble de dicho tamaño, si lo que deseamos es minimizar el tiempo que tardan dichas operaciones, nos inclinaremos por tamaños de páginas Grandes. Por otro lado ,un tamaño de página Más pequeño mejora la localidad de un proceso , permitiendo que cada página se ajuste precisamente con la misma, y no sea necesario cargar porciones de programa en memoria que no se

utilicen realmente durante su ejecución , lo que reduciremos así es la cantidad de operaciones de E/S y la cantidad total de memoria asignada .Pero al utilizarse una página más pequeña ,también aumentará la cantidad de fallos de página y por ende se reducirá la utilización de la CPU, por lo tanto, para minimizar el número de fallo de pagina, nos inclinaremos por páginas de mayor tamaño. A pesar de todo lo dicho, la tendencia histórica es hacia páginas De tamaño más grande.

- ✓ Almacén en TLB :
  Como dijimos antes, la tasa de aciertos de la TLB, (es decir el porcentaje de traducciones de direcciones virtuales que se resuelven mediante el TLB y no mediante la tabla de páginas). Está relacionada claramente con el número de entradas contenidas en la TLB, de manera que si se incrementa la cantidad de entradas, se incrementa la tasa de aciertos. Otra métrica relacionada con la tasa de aciertos es el "Alcance de la TLB", que hace referencia a la cantidad de memoria accesible a partir de la TLB, es decir el número de entradas multiplicado por el tamaño de pagina. Idealmente la TLB podrá almacenar el conjunto de trabajo completo de un proceso, de no ser así se desperdiciara un tiempo valioso traduciendo las referencias a memoria mediante la tabla de páginas. Una técnica para lograr incrementar el alcance de la TLB consiste en incrementar el tamaño de la página o proporcionar tamaños múltiples. Proporcionar soporte para tamaños múltiples de página requiere que sea el sistema operativo (y no el hardware) el que gestione la TLB.
- ✓ Tabla de páginas invertida:
  Por su estructura la tabla de paginas invertida no contiene información completa a cerca del espacio lógico de direcciones de un proceso y dicha información es necesaria si una página a la que se haga referencia no se encuentre en memoria, entonces debemos almacenar una tabla de paginas externa por cada proceso. Cada una de estas tablas es similar a la tabla de páginas tradicional de un proceso y contiene información sobre la ubicación de cada proceso. Estas tablas solo se consultan en caso de fallo de pagina y se las puede cargar y descargar de memoria según sea necesario, esto hace que todas las ventajas adquiridas por la tabla de paginas invertida no se pierdan en estas tablas, sin embargo hay casos en los que puede que un fallo de pagina común produzca otro fallo de pagina para cargar alguna de estas tablas.
- ✓ Estructura de programas:
  Como dijimos la paginación bajo demanda está diseñada para ser transparente al programa de usuario, pero en ocasiones puede que si el usuario (o el compilador) conoce la naturaleza paginada de la memoria, mejore el rendimiento, construyendo programas más acorde a las características del sistema en el que va a ejecutar.
- ✓ Interbloqueo de E/S:
  Cuando se utiliza paginación bajo demanda, en ocasiones es necesario permitir que ciertas páginas queden bloqueadas en memoria. A modo de ejemplificación, ilustraremos una de ellas; si un proceso realiza una solicitud de E/S y es puesto en la cola de dicho dispositivo, y mientras tanto la CPU se asigna a otro proceso, el cual provoca un fallo de pagina y el algoritmo de sustitución global sustituye la pagina que contiene el buffer de memoria del proceso en espera, se producirá un problema al momento en que la E/S finalice y descubra que el marco donde tiene que almacenar los datos está ocupado por otro proceso. Como ya se dijo, la solución más simple consiste en no ejecutar nunca operaciones de E/S sobre la

memoria de usuario, y en lugar de ellos copiar siempre en la memoria del kernel primero y luego de la del mismo a la del usuario cuando sea propicio. Existe otra alternativa que no es tan inaceptablemente costosa y es la de bloquear un marco como antes se dijo. Para implementar el bloqueo se utiliza un bit de bloqueo asociado a cada marco, si este no está activado no podrá ser seleccionado para sustitución. Este bit se utiliza de forma usual para bloquear alguno o todos los marcos pertenecientes al kernel.

Capitulo 10:
Interfaz del Sistema de Archivos

El sistema de archivos (File System), está compuesto básicamente por tres elementos, una colección de archivos, una estructura de directorios que los organiza, e información acerca de los mismos. Una de las funciones principales de la interfaz del sistema de archivos es proveer abstracción al usuario, con respecto a los dispositivos de almacenamiento que manipula el sistema de archivos propiamente dicho, llegando así a que para el usuario el sistema de archivos sea concebido como un sistema uniforme. El máximo exponente de dicha abstracción es lo que dimos a llamar anteriormente como "Archivo", y así entonces llegamos a su definición:

Archivo: unidad básica de almacenamiento lógico secundario del usuario, compuesta por un conjunto de datos relacionados cuyo significado se encuentra definido por el creador del mismo o el que lo utilice, que tienen un conjunto de atributos.

Estos archivos están mapeados en los diferentes dispositivos de almacenamiento, y como es evidente son de carácter no volátil.
Como dijimos en la definición anterior de archivo, los archivos tienen un conjunto de atributos, los cuales varían de un sistema operativo a otro pero en general podemos enumerar los siguientes básicos que todos comparten:

- ✓ Nombre: un conjunto de caracteres que denotan el nombre del archivo, el único atributo legible.
- ✓ Identificador: una etiqueta univoca que identifica el archivo en el sistema de archivos, generalmente un número.
- ✓ Tipo: es necesario para los sistemas operativos que soporten tipos de archivos (ejemplo: Windows).
- ✓ Ubicación: un puntero a un dispositivo y a la ubicación del archivo dentro de dicho dispositivo.
- ✓ Tamaño: tamaño actual del archivo en bytes, palabras o bloques y posiblemente el tamaño máximo permitido.
- ✓ Protección: información de control de acceso.
- ✓ Fecha, hora e identificación de usuario: puede guardarse por creación, modificación o último acceso.

La información acerca de los archivos, se almacena en la estructura de los directorios, de forma general una entrada a directorio está compuesta por los atributos "Nombre" e "Identificador", el segundo permite localizar los atributos restantes del archivo.
Existen una serie de operaciones con archivos que el sistema operativo pone a disposición del usuario ellas son las que siguen:

- ✓ Crear archivo:
  - Encontrar espacio disponible.
  - Incluir en el directorio una entrada para el nuevo archivo.
- ✓ Escritura de un archivo:
  - Obtener el nombre del archivo y datos a escribir.
  - Ubicar el archivo mediante el nombre en la estructura de directorios.

- Mantener un puntero que haga referencia a una ubicación para la escritura.
- Actualizar el puntero por cada escritura de datos realizada en el archivo.

✓ Lectura de un archivo:
- Obtener el nombre del archivo.
- Ubicar el archivo mediante el nombre en la estructura de directorios.
- Cargar el archivo en la memoria principal.
- Mantener un puntero que haga referencia a una ubicación para la lectura.
- Actualizar el puntero por cada lectura de datos realizada en el archivo.

✓ Reposicionamiento dentro de un archivo:
- Ubicar el archivo en la estructura de directorios.
- Actualizar el puntero de posición actual dentro del archivo asignándole un nuevo valor.

✓ Borrado de un archivo:
- Obtener el nombre del archivo.
- Ubicar el archivo mediante el nombre en la estructura de directorios.
- Liberar el espacio asignado al archivo para que pueda ser utilizado por otros archivos.
- Borrar la entrada de directorio correspondiente al archivo.

✓ Truncado de un archivo:
- Obtener el nombre del archivo.
- Ubicar el archivo mediante el nombre en la estructura de directorios.
- Modificar el atributo "Tamaño" otorgándole el valor "0" y manteniendo el resto de los atributos intactos.

Nota: el puntero de lectura, escritura, y reposicionamiento es el mismo para no utilizar más memoria sin no hay necesidad.

Aclaración: se pueden agregar también las operaciones de "Adición" (permite agregar información al final del archivo) y "Renombrado" (permite cambiar el valor del atributo "Nombre" del archivo).

Estas son las operaciones más básicas y generales que el sistema operativo le provee al usuario, el resto de las operaciones se construyen con la combinación de algunas de ellas.
Dado que la mayor parte de dichas operaciones básicas, tienen como acción necesaria la obtención del atributo "Nombre" del archivo, y luego mediante dicho atributo la búsqueda de su respectiva entrada de directorio en la estructura de directorios, tiene sentido hablar de la primitiva "Open()", que se encargaría de dicha tarea, es por esto que en muchos sistemas operativos esta primitiva es invocada antes de utilizar un archivo, nos provee de la idea de que para utilizar un archivo primero hay que "Abrirlo" y si se abrió, en algún momento luego de utilizarlo deberíamos "Cerrarlo" objetivo de la primitiva "Close()", cabra mencionar otras también llegada la ocasión, pero antes de hacerlo, es necesario comentar a cerca de la existencia de los dos niveles de tablas de archivos abiertos en memoria principal que manipula el sistema, y la razón de su existencia:

Tabla Global de Archivos Abiertos: la razón de su existencia es trivial, si más de un proceso abre un mismo archivo se desperdiciaría memoria en mantener la misma

entrada de directorio replicada la cantidad de veces que dicho archivo sea abierto. Para solucionar este problema se define esta tabla con una única entrada por cada archivo distinto almacenado que haya sido abierto, con los atributos de carácter global de la apertura del archivo, es decir los que no tienen que ver con cuestiones particulares de cada apertura de proceso, sino con aquellas que comparten todas las aperturas del mismo archivo, algunas de ellas a modo de ilustración podrían ser:

- ✓ Contador de Aperturas: se mantiene un numero que representa la cantidad de aperturas existentes del archivo en un instante, este atributo se incrementa en uno cuando se llama a la primitiva "Open()" y se decrementa en uno cuando se llama a la primitiva "Close()", la razón de este atributo es básicamente que cuando su valor se igual a "0" se elimina la entrada de la tabla.
- ✓ Ubicación: el mismo atributo de archivo que contiene la entrada de directorio, se repite dado que sino, por cada modificación de datos dentro del archivo se tendría que extraer este atributo de dicha entrada, ralentizando la operación, ya que esta tabla se halla en memoria principal y la entrada de directorio generalmente en disco.

Tabla de Archivos Abiertos por proceso: la razón de su existencia también es trivial si se tiene en cuenta que como se dijo antes la anterior tabla es global, y por ende no almacena los atributos particulares de cada apertura de un mismo archivo, estos se encuentran disgregados por todas estas tablas, las cuales se distribuyen de a una por proceso conteniendo una entrada por cada archivo abierto por ese proceso, también a modo de ilustración podemos enumerar algunos de estos atributos compartidos por todos los archivos abiertos por un mismo proceso:

- ✓ Puntero de Archivo: un puntero a la posición actual dentro del archivo, su existencia radica en que el proceso necesita conocer la posición actual en el archivo con la que está trabajando, su valor se actualiza con cada lectura, escritura o reposicionamiento dentro del archivo.
- ✓ Puntero a la "Tabla Global de Archivos Abiertos": como el nombre lo dice es un puntero a la entrada correspondiente al archivo abierto en la tabla global, la razón de este atributo, es que los datos globales del archivo abierto por el proceso no se encuentran en esta tabla por ende se necesita saber dónde buscarlos.
- ✓ Derechos de Acceso: se mantiene el modo de apertura del archivo, con el objetivo de que el sistema operativo puede autorizar o denegar las operaciones que el proceso le solicite realizar con el archivo.

Realizado el comentario anterior podemos enumerar las acciones de la primitiva "Open()" como sigue:

- ✓ Obtener el nombre del archivo.
- ✓ Obtener el modo de apertura del archivo (ejemplo: creación, solo lectura, solo escritura, etc.).
- ✓ Ubicar el archivo mediante el nombre en la estructura de directorios.
- ✓ Comparar el modo de apertura del archivo con el o los valores del atributo "Protección" del archivo.
- ✓ Dependiendo de la operación anterior permitir o no al proceso la apertura del archivo.

- ✓ Verificar si es la primera vez que se abre el archivo buscando en la "Tabla Global de Archivos Abiertos" una entrada con el mismo atributo "Nombre".
- ✓ Dependiendo de la operación anterior generar una nueva entrada en la "Tabla Global de Archivos Abiertos" del sistema, copiando en ella parte de la entrada de directorio del archivo, e inicializando en "1" el valor del atributo "Contador de Aperturas" de dicha entrada, luego generar una nueva entrada en la "Tabla de Archivos Abiertos" del proceso; esto sucede si es la primera vez que se abre el archivo, de no darse de esta forma solamente se genera una entrada en la última tabla, y en la global se incrementa en uno el valor del atributo "Contador de Aperturas".
- ✓ Sin depender del resultado de la operación anterior se devuelve al proceso un puntero a la entrada correspondiente dentro de su "Tabla de Archivos Abiertos", con el que se realizaran el resto de las operaciones, comúnmente se lo denomina "Descriptor de Archivo" (File Descriptor).

Y para la primitiva "Close()":

- ✓ Obtener el "Descriptor de Archivo".
- ✓ Decrementar en uno el atributo "Contador de Aperturas" de la entrada correspondiente al archivo de la "Tabla Global de Archivos Abiertos" del sistema.
- ✓ Dependiendo de la operación anterior, si el atributo "Contador de Aperturas", luego del decremento adquirió el valor "0", se actualizan los atributos de la entrada de directorio del archivo con los correspondientes a la entrada de la tabla global, y luego se elimina esta última entrada.
- ✓ Sin depender del resultado de la operación anterior se elimina el "Descriptor de Archivo" de la "Tabla de Archivos Abiertos" del proceso.

Existen otras primitivas como "Read()", "Write()", "Seek()", "Create()", "Delete()", etc., cuyas implementaciones asumiremos que son como las de sus análogas mencionadas en la sección anterior cundo se enumero las operaciones con archivos que el sistema operativo le provee al usuario.

Tipo de Archivo:
Puede verse a los "tipos de archivos" como una forma de clasificar a los archivos según su contenido, dicha clasificación le sirve al usuario o al sistema operativo, no todos los sistemas operativos implementan tipos de archivos, pero les permite a aquellos que si lo hacen operar de forma razonable con los archivos, es decir sin intentar operar con el archivo de una forma que no sea acorde al significado del contenido del archivo. Una posible forma de implementar los "tipos de archivos" es adicionando al atributo "Nombre" de un archivo una serie de caracteres extra, para identificar el "tipo" del archivo, a este conjunto extra de caracteres se los denomina "extensión", y por lo general se adición al final de la cadena que representa el nombre del archivo un punto y luego la "extensión". Otra forma es la de utilizar un atributo de archivo "Tipo" que lo especifique, como se menciono en la sección de atributos de un archivo.

Estructura de los archivos:
Como mencionamos anteriormente los "tipos de archivos" son una forma de clasificar los archivos según su contenido, el cual tiene una estructura determinada, que es necesaria para poder "interpretarlos", si se conoce el "tipo de archivo" se puede saber

que programas son candidatos para “interpretarlos”; serán aquellos que conozcan la estructura asociada al tipo de archivo que se pretende interpretar. Hay algunas estructuras particulares de archivos que son soportadas por los sistemas operativos, dado que los mismos necesitan operar con dichos archivos, y para hacerlo han de conocer sus estructuras. Atendiendo a esta última cuestión no todos los sistemas operativos cumplen con el lineamiento que sigue:
“Cada archivo es una secuencia de bytes de 8 bits sin que el sistema operativo realice ninguna interpretación de dichos bits”.
De hecho ninguno cumple con esto dado que cualquier sistema operativo debe conocer la estructura de un archivo ejecutable, y al conocimiento de esta podrán o no adicionarse otras estructuras de archivos pero en esencia como mínimo la de archivo ejecutable. Mientras más estructuras soporte el sistema operativo mas excesivamente grande será, pero más fácil la tarea del programador, dado que no es necesario que sus aplicaciones conozcan la estructura de aquellos archivos que soporta el sistema operativo.

Estructura interna de los archivos:
Localizar un desplazamiento dentro de un archivo puede ser una tarea ardua para el sistema operativo, dado que los almacenamientos secundarios como el disco tienen registros físicos fijos como los sectores, y en contraposición los registros lógicos de datos no solo puede que no tengan el mismo tamaño, sino que también en ocasiones varían su tamaño de operación en operación. La solución a este problema la aporta el concepto de “bloque”, en uno de ellos se reúnen múltiples registros lógicos, tantos como entren en un tamaño fijo que se le atribuye al mismo. Entonces siguiendo con esta idea podemos pensar en un archivo como un conjunto de “bloques” cada uno de ellos de un tamaño fijo, y para realizar operaciones con el byte i-esimo del archivo deberemos empaquetar y desempaquetar continuamente los bloques del mismo, tarea que se le otorga al sistema de archivos. El único efecto negativo de la división en bloques es el que se da a conocer como “Fragmentación Interna”.

Fragmentación Interna: este fenómeno se da por el simple hecho de pensar un archivo como un conjunto n-esimo de bloques de tamaño fijo como se dijo anteriormente, dado que si en el archivo se almacenan registros lógicos medidos en cantidades de bytes que pueden variar de un registro lógico a otro y que además no necesariamente coinciden con el tamaño del bloque, causara que el n-esimo bloque de un archivo este siempre incompleto y que solamente en algunos casos excepcionales este completo, esto produce el desperdicio de un conjunto de bytes. Todos los sistemas de archivos padecen este fenómeno, y mientras más grande el tamaño del bloque mayor el efecto del fenómeno.

Métodos de acceso:
A pesar de que varían de un sistema operativo a otro, los más utilizados son:

- ✓ Acceso secuencial: la información del archivo se procesa en orden, un registro después de otro. Para lecturas se lee la poción que indica el “Puntero de Archivo” del descriptor de archivo, y luego se incrementa el valor del mismo. Para escrituras se añade información al final del archivo y se obliga al “Puntero de Archivo” a avanzar hasta la posición del nuevo fin de archivo. Está basado en el modelo de archivo para cintas magnéticas.
- ✓ Acceso directo: el archivo se divide en una cantidad n-esima de registros de igual tamaño, y se accede al número de registro deseado, luego se lee o escribe

en el sin necesidad sin restricción de orden de lectura o escritura. Está basado en el modelo de archivo de disco.

- ✓ Acceso indexado: se construye por encima del método de acceso directo, otorgando un índice para el archivo, este contiene punteros a los distintos registros, si se desea realizar una operación de lectura o de escritura de un registro solamente se accede al puntero indicado en el índice.

Estructura de directorios:
Un dispositivo de almacenamiento secundario puede utilizarse completamente bajo un solo sistema de archivos, o puede tener múltiples sistemas de archivos, en este último caso hablamos de particiones. Estas particiones a su vez pueden combinarse para formar otro sistema de archivos mas abarcativo denominado volumen, cada uno de estos volúmenes puede ser considerado un disco virtual. El volumen ha de conocer la información inherente a cada uno de los archivos almacenado en su sistema de archivos, esta información se encuentra en las entradas de directorio de la estructura de directorios del volumen. Cada directorio de la estructura de directorios puede ser visto como aquel encargado de relacionar un nombre de archivo, con su correspondiente entrada de directorio. Es preciso definir el conjunto de operaciones que podemos realizar con un directorio, antes de abordar los diferentes esquemas posibles para la estructura de directorios:

- ✓ Búsqueda de un archivo: se explora la estructura de directorios en busca de la entrada de directorio correspondiente al archivo que se desea encontrar, según una referencia.
- ✓ Creación de un archivo: se genera una entrada adicional en la estructura de directorios para albergar al nuevo archivo.
- ✓ Borrado de un archivo: se borra la entrada de directorio correspondiente al archivo que se desea eliminar.
- ✓ Listar un directorio: se recorre el directorio entrada por entrada listando los atributos de cada archivo.
- ✓ Renombrar un archivo: modificar el atributo “Nombre” de la entrada de directorio indicada por el usuario, o mover dicho archivo a otra entrada de directorio dentro de la estructura de directorios
- ✓ Recorrer el sistema de archivos: acceso a todos los directorios y todos los archivos contenidos en la estructura de directorios.

Teniendo en cuenta las operaciones anteriores podemos analizar los diferentes esquemas para la estructura de directorios:

- ✓ Directorio de un único nivel: en este esquema existe un solo directorio en todo el sistema de archivos que contiene todos los archivos existentes. Las principales desventajas de este esquema son si se incrementa en demasía el número de archivos, con consecuencias como que el usuario no pueda recordar los nombres de todos los archivos, y si hay más de un usuario en el sistema
- ✓ Directorio de dos niveles: en este esquema existe un primer nivel conformado por un único directorio, cuyas entradas hacen referencia a cada uno de los usuarios en el sistema, y se da a llamar “Directorio Maestro de Archivos” (Master File Directory), y un segundo nivel que se compone de una serie de directorios de a uno por cada usuario del sistema llamado “Directorio de Archivos de Usuario” (User File Directory), cada una de sus entradas representa

un archivo perteneciente a un mismo usuario. Este esquema resuelve el problema anterior con respecto a los usuarios, dado que por ejemplo costara menos mantener la unicidad de los nombres de los archivos por cada usuario que en todo el sistema, y además si los usuarios fueren muy independientes entre sí, efectivamente el sistema lograría mantenerlos aislados unos de otros, pero es esta ventaja también su principal falencia en casos en los que los usuarios deseen cooperar entre si y compartir archivos, y cuando así acontezca dependerá de cómo resuelva la situación cada sistema operativo distinto, dejando o no a un usuario ver los archivos del resto de los usuarios. En este esquema es donde se comienza con la idea de tener siempre un usuario adicional, "usuario 0", con su correspondiente entrada de directorio en el "Directorio Maestro de Archivos", para albergar los archivos del sistema.

- ✓ Directorios con estructura de árbol: este esquema es la generalización del anterior para "n" niveles, se tendrá un único directorio en el primer nivel denominado "raíz" y luego tantos niveles como se desee que tenga el árbol. Una de las ventajas principales de este esquema es que permite a los usuarios generar sus propios directorios y organizar sus archivos. Directorios y archivos poseen la misma estructura interna, la única diferencia es un bit que en cada entrada de directorio está en "0" si se trata de un archivo y en "1" si se trata de un directorio. En este esquema en casi todos los sistemas operativos se le permite a un usuario acceder a los archivos de otros usuarios, además de a suyos.
- ✓ Directorios en un grafo acíclico: este esquema es más general aun que el anterior, agregándole al mismo la posibilidad de que las entradas de directorio distintas puedan compartir un mismo directorio o archivo. La implementación de este esquema de se puede realizar a modo de ejemplificación, con una entrada de directorio que posea un atributo especial que indica que funciona como enlace (link) entre él y el directorio o archivo a compartir, es decir una entrada de directorio que sea un puntero al directorio o archivo a compartir. Una de las principales desventajas de este esquema es la replicación de la información inherente a un mismo archivo o directorio, tomando como ejemplo la ruta para acceder a un directorio o archivo, siendo este compartido habrá más de una ruta de acceso al mismo, y eso cobra vital importancia a la hora de realizar una exploración completa del sistema de archivos, dado que sería deseable no recorrer más de una vez la misma sección de la estructura de directorios. Otro problema tiene que ver con el borrado de los archivos, dado que si se liberase el espacio asignado a un archivo compartido inmediatamente luego de solicitar el borrado del mismo, podríamos correr el riesgo de que quedasen punteros colgantes, dado que una vez vuelto a asignar ese espacio estos punteros terminarían apuntando a otros archivos o partes de ellos. Una posible solución a este último problema es; si se solicita el borrado de un enlace o de la misma entrada de directorio del archivo compartido, se eliminan ambas, y si en el segundo caso quedasen punteros colgantes se ignoraran hasta el momento en que se quiera acceder mediante ellos al archivo compartido, en ese momento el sistema operativo le indicara al usuario que no pudo resolver la ruta especificada, y lo que suceda con el enlace se deja a criterio del usuario. Otra solución posible seria mantener una lista que por cada elemento tuviese una referencia al archivo compartido y un numero de enlaces al mismo existentes en cada instante, entonces si se añade un nuevo enlace se incrementara en uno dicho valor y si se elimina la inversa, al momento en que el valor sea "0" se libera el espacio asignado al archivo.

- ✓ Directorio en forma de grafo general: nuevamente este esquema es mas general que el anterior, porque además nos permite armar subestructuras de directorios cíclicas, lo cual propone una dificultad mayor que en el esquema anterior, para evitar volver a recorrer las mismas secciones de la estructura de directorios más de una vez en una búsqueda global en el sistema de archivos, posible solución consiste en limitar arbitrariamente el numero de directorios a los que se accederá durante una búsqueda. De forma similar también al esquema anterior se pueden producir problemas al intentar borrar un archivo compartido, dado que la solución anterior de tener un contador por la cantidad de enlaces a un archivo compartido ya no será válida si por ejemplo existe un auto enlace, en ese caso el contador de enlaces tendría un valor distinto de "0" y sin embargo ya no se podría acceder al archivo compartido. Este último problema se podría resolver mediante una técnica de recolección de memoria, la cual consiste en realizar dos recorridos por todo el sistema de archivos, la primera identificando los elementos a los cuales se puede acceder, la segunda liberando el espacio asignado a aquellos elementos no identificados en la primera recorrida, sin embargo es una técnica que consume ingentes cantidades de tiempo cuando se trata de un dispositivo como el disco, es por esto que se suele escoger el esquema anterior.

Algunas definiciones adicionales:

Ruta de Búsqueda: se define así a la secuencia de directorios que se exploran cada vez que se proporciona un nombre de archivo.

Directorio Actual: se le asigna a cada proceso, y representa el directorio en el cual se localiza actualmente dicho proceso. El valor del directorio actual puede modificarse mediante la llamada al sistema indicada, inicialmente cada proceso posee como directorio actual el de su proceso padre al momento de crearlo.

Nombre de Ruta Absoluta: comienza con la raíz y termina con el nombre del archivo, y en el espacio entre ambos están todos los directorios que se deben recorrer desde la raíz hacia los niveles inferiores hasta llegar al archivo.

Nombre de Ruta Relativo: define una ruta a partir del "Directorio Actual".

Montaje de sistemas de archivo:
Un sistema de archivos debe montarse para estar disponible para los procesos. El proceso de montaje es sencillo como se lo detalla a continuación:

- ✓ El usuario le solicita montar un dispositivo al sistema operativo, indicándole a este el nombre del dispositivo y el punto de montaje, normalmente este punto será un directorio vacío.
- ✓ El sistema operativo verifica que el dispositivo especificado por el usuario tenga un sistema de archivos que este puede interpretar, solicitándole al controlador del dispositivo que lea el directorio y verifique si tiene el formato indicado.
- ✓ El sistema operativo registra en su estructura de directorios que en el "Punto de Montaje" hay sistema de archivos montado, esto le permite al mismo recorrer su estructura de directorios conmutando de un sistema de archivos a otro si fuese necesario.

- ✓ Finalmente el usuario puede acceder al directorio como si fuera cualquier otro de la estructura de directorios.

Punto de Montaje: se le da esta denominación a una ubicación cualquiera dentro de la estructura de archivos a la cual se conectara el sistema de archivos que se montara.

Compartición de Archivos:
En este tipo de cuestiones el sistema operativo debe adoptar un papel de mediador, puede este permitir que los usuarios accedan a los archivos de otros usuarios de manera predeterminada, o puede por el contrario requerir que los usuarios concedan explícitamente el acceso sus archivos. Para implementar la compartición y la protección de archivos es sistema debe reforzar la cantidad de atributos de los archivos.
Definiremos dos conceptos necesarios para lograr la compartición de archivos:

Propietario: el usuario que posee del máximo grado de control sobre un archivo o directorio, pudiendo este cambiar sus atributos y conceder accesos a él.

Grupo: un subconjunto de usuarios que pueden compartir el acceso a un archivo o directorio.

Los identificadores del propietario y del grupo de un archivo o directorio se almacenan junto con los otros atributos del archivo, de este modo es sistema operativo puede según quien le solicita realizar una operación con dicho archivo o directorio indicar que permisos le son aplicables, y autorizar o denegar la operación.
Sistemas de archivos remotos: existen tres métodos de compartición de archivos remotos estos son:

- ✓ Una transferencia manual mediante protocolos como “ftp” (File Transfer Protocol).
- ✓ Un “DFS” (Distributed File System), en el que los directorios remotos son visibles desde una maquina local.
- ✓ La “www” (World Wide Web), un explorador para poder acceder a los archivos remotos y se utilizan operaciones independientes para transferir los archivos.

Existen dos tipos de accesos a los archivos remotos:

Acceso Anónimo: permite al usuario transferir archivos sin disponer de una cuenta en el sistema remoto. De los tres mecanismos anteriores en algunos casos “ftp”, y la “www”.

Acceso Autenticado: impide al usuario transferir archivos sin antes haberse autenticado en el sistema remoto. De los tres mecanismos anteriores en algunos casos “ftp”, y “DFS”.

Semántica de coherencia: esta específica como pueden acceder simultáneamente a un archivo los múltiples usuarios del sistema, en particular especifica cuando las modificaciones que un usuario realice en los datos serán observables por parte de los otros usuarios.

A modo de ejemplificación del concepto de semántica de coherencia tomaremos el caso de los archivos compartido inmutables. Una vez que un archivo en esta semántica es declarado compartido por su creador, no puede ser modificado; su nombre no puede reutilizarse, y su contenido no puede ser modificado, por ende se lo denomina "Inmutable", dado que su contenido es fijo.

Protección:
Cuando se almacena información en un sistema informático, uno de los aspectos de la protección consiste en protegerla frente a los accesos incorrectos (protección), la primera será tratada con más detalle en el capítulo 12 y en el caso de la segunda, veremos a continuación algunos casos.
Tipos de acceso: se podría plantear un esquema en donde hubiese protección total de los archivos o directorios, simplemente prohibiendo todo acceso de un usuario a los archivos o directorios de otro, como también podríamos plantear un esquema donde no hubiese protección alguna, permitiendo el acceso libre sin ninguna protección, pero ambos esquemas son muy radicales, por ende lo que se necesita es un acceso controlado. La forma de implementar este tipo de acceso es limitar los tipos de accesos al archivo que puedan realizarse, estos son:

- ✓ Lectura: se lee el contenido de un archivo.
- ✓ Escritura: se escribe o reescribe el contenido de un archivo.
- ✓ Ejecución: se carga el archivo en memoria y se ejecuta el mismo.
- ✓ Adición: se escribe nueva información al final del archivo.
- ✓ Borrado: se borra el archivo y se libera el espacio para su posible reutilización.

Enumeraremos a continuación una serie de técnicas de protección:

- ✓ Control de acceso: se asocia a cada archivo o directorio una "Lista de Control de Acceso" (ACL, access-control list), donde se enumeran todos los nombres de usuario y los tipos de acceso que se permiten a para cada uno. La ventaja de este esquema de protección, consiste en la complejidad de las metodologías de acceso, y sus desventajas son básicamente, la longitud de esta lista, la forma de construcción de la misma dado que no se sabe de antemano cuantos usuarios tiene el sistema, y la entrada de directorio deberá ser de tamaño variable. Para mejorar este tipo de desventajas se opta por condensar la lista en tres tipos de clasificación de usuarios con respecto a cada archivo:
    - Propietario: el usuario que creó el archivo.
    - Grupo: un conjunto de usuarios que están compartiendo el archivo y necesitan acceder de forma similar al mismo.
    - Universo: todos los demás usuarios del sistema que no pertenecen a las dos primeras clasificaciones.

  La forma de implementar este esquema, es mediante tres atributos para definir los mecanismos de protección, cada campo es a menudo una colección de bits. A modo de ejemplificación podemos considerar el sistema UNIX, donde estos tres campos tienen tres bits cada uno; el llamado "r" (read – lectura), el llamado "w" (write – escritura), y el llamado "x" (execute – ejecución), y la presencia o ausencia de los mismos en cada campo determina los permisos. Adicionalmente se le puede añadir a la lista condensada una ACL más pequeña que gestione los casos particulares, de haber tal lista el sistema operativo deberá encargarse de resolver las superposiciones entre los permisos y la ACL de un mismo archivo,

por ejemplo resolviendo por esta última con el criterio de “lo más particular pesa más que lo mas general”.

- ✓ Acceso por contraseña: se asigna a cada archivo o directorio compartido una contraseña, si se eligen estas de forma aleatoria y se cambian con frecuencia, puede ser un esquema de protección muy efectivo. Sin embargo, el uso de contraseñas tiene unas cuantas desventajas, como el número de contraseñas que el usuario deberá retener, y si se utiliza la misma contraseña para todos los archivos una vez descifrada esta, se podrá acceder a todos los archivos del usuario.

Las operaciones a proteger en los directorios son algo diferentes que las operaciones a proteger en los archivos, la más importante de estas a proteger es, listar un directorio, dado que mediante los nombres de los archivos listados, los usuarios confeccionaran las rutas de esos archivos, para accederlos, y ese tipo de operación no estará protegido. En aquellos casos donde un mismo archivo tenga más de una ruta (estructura de directorios como grafo acíclico o general), un usuario puede tener diferentes derechos de acceso a un archivo concreto, dependiendo del nombre de ruta que utilice.

Capitulo 11:
Implementación de Sistemas de Archivos

Los discos son el principal tipo de almacenamiento secundario para mantener sistemas de archivos, porque:

- ✓ Un disco puede ser reescrito de forma directa, se puede leer un bloque, modificarlo, y volverlo a escribir en el mismo lugar.
- ✓ En un disco se puede acceder directamente a cualquier bloque de información que contenga.

En vez de transferir de a un byte, las transferencias entre la memoria y el disco se hacen en unidades de bloques, donde cada bloque tiene uno o más sectores (en cuyo caso se llaman cluster). Generalmente el tamaño del bloque es 512 bytes, pero varían entre 32 bytes y 4 kilobytes.

Niveles de diseño del sistema de archivos:
Desde el más bajo al más alto serian:

- ✓ Control de E/S: compuesto por controladores de dispositivo y rutinas de tratamiento de interrupciones, que transfieren la información entre la memoria principal y el disco. Un controlador de dispositivo (también llamado driver, a nivel software) es un traductor, su entrada son cadenas de alto nivel (ejemplo: “extraer bloque 17”), y su salida son instrucciones de bajo nivel especificas del hardware que son usadas por la controladora de hardware (nivel hardware, especificas de cada dispositivo).
- ✓ Sistema básico de archivos: envía comandos genéricos al controlador de dispositivo, para poder leer y escribir bloques físicos en el disco (ejemplo: “escribir Unidad 1, cilindro 4, cabezal 2, sector 9”).
- ✓ Modulo de organización de archivos: conoce los archivos y sus bloques lógicos, los cuales convierte a direcciones físicas y los manda al sistema básico de archivos para que realice las transferencias. También contiene al gestor de espacio libre, que controla los bloques no asignados y los proporciona al modulo de organización de archivos cuando este lo solicita.
- ✓ Sistema lógico de archivos: gestiona la información de meta datos (toda la estructura del sistema de archivos, excepto los propios datos), para darle al modulo de organización de archivos la información que necesita a partir del nombre de un archivo.
- ✓ Esta estructura de archivos se mantiene mediante bloques de control de archivo (FCB, File Control Block), los cuales tienen la información acerca del archivo (propietario, permisos, ubicación del contenido, etc.).

Implementación del sistema de archivos:
Para implementar un sistema de archivos, se usan varias estructuras en disco y en memoria. En disco se tiene la información sobre cómo iniciar un sistema operativo, el número total de bloques, el número y ubicación de los bloques libres, las estructuras de directorios y archivos.

- ✓ Bloque de control de arranque: existe uno por volumen, con la información necesaria para iniciar el sistema operativo a partir de dicho volumen. Si el disco

no tiene un sistema operativo, queda vació. Es el primer bloque del volumen. En UFS se llama bloque de inicio, y en NTFS, sector de arranque de la partición.

- ✓ Bloque de control de volumen: existe uno por volumen, con información sobre el numero de bloques, su tamaño, el numero de bloques libres y punteros a los mismo, un contador de FCB libres y punteros a los mismo. En UFS se llama superbloque, en NTFS, tabla maestra de archivos.
- ✓ Estructura de directorios por cada sistema de archivos, para hacer los archivos. En UFS tiene los nombres de archivo y los nombres de inodo asociados, en NTFS esta información está también en la tabla maestra de archivos.

En memoria las estructuras se cargan en el momento del montaje y se borran cuando se desmonta el dispositivo:

- ✓ Tabla de montaje: información sobre cada volumen montado.
- ✓ Cache de la estructura de directorios: información sobre los directorios accedidos recientemente.
- ✓ Tabla global de archivos abiertos: tiene una copia del FCB de cada archivo abierto más otra información.
- ✓ Tabla de archivos abiertos de cada proceso: tiene un puntero a la entrada correspondiente al archivo en la tabla global de archivos abiertos más otra información.

Creación de un nuevo archivo:
Para crear un archivo, el programa de aplicación llama al sistema lógico de archivos, quien conoce el formato de las estructuras de directorio, y asigna un FCB libre para el archivo, lee el directorio apropiado en la memoria, lo actualiza con el nombre de archivo nuevo y su FCB, y lo vuelve a escribir en el disco.

Abrir un archivo:
Open() le pasa un nombre de archivo al file system, quién verifica en la tabla global de archivos abiertos si el archivo ya fue abierto por otro proceso. Si está abierto, crea una entrada en la tabla de archivos abiertos del proceso que apunte a la entrada correspondiente en la tabla global, y guarda más información como por ejemplo el puntero a la posición actual dentro del archivo, y el modo de acceso en que se abrió el archivo (lectura, escritura, etc.). Si no está abierto, se busca el archivo en la estructura de directorios mediante su nombre, una vez encontrado, se copia el FCB en la tabla global de archivos abiertos que también va a tener un contador de aperturas del archivo. Open() devuelve un puntero a la entrada del archivo en la tabla de archivos abiertos del proceso. Todas las operaciones sobre el archivo se van a realizar mediante ese puntero. El nombre del archivo puede no estar en la tabla de archivos abiertos por proceso ya que no se usa porque se tiene su FCB. El nombre cada entrada de la tabla de archivos abiertos por proceso varía según el sistema, en UNIX se llama file descriptor, y en Windows se llama file andel. Al abrir un archivo, se carga en la caché de estructuras de directorio toda la información a cerca de dicho archivo, salvo los propios bloques de datos, para minimizar la cantidad de operaciones de E/S al disco.

Cerrar un archivo:
Cuando un proceso llama a Close() para cerrar un archivo, se elimina la entrada de la tabla de archivos abiertos del proceso, y se decrementa el contador de aperturas en la tabla global de archivos abiertos. Cuando dicho contador llega a cero (nadie lo está

usando), los meta datos actualizados se copian en la estructura de directorio almacenada en el disco, y se elimina la entrada de la tabla global de archivos abiertos.

Métodos de asignación:

Asignación contigua:
Cada archivo ocupa un conjunto de bloques contiguos en el disco. Los movimientos de cabezal para cada acceso secuencial o aleatorio son los mismos. La asignación contigua está definida por la asignación de disco del primer bloque y por la longitud del archivo (en unidades de bloques). Si el archivo arranca en la posición "x" y tiene "n" bloques de longitud entonces ocupara en disco desde el bloque "x" hasta el "x+n-1". La entrada de directorio de cada archivo indica la dirección del bloque de inicio y la longitud del área asignada al archivo. Para acceso secuencial, el sistema de archivos recuerda la dirección de disco del último boque al que se haya hecho referencia y leerá el siguiente bloque cuando sea necesario. Para acceso directo al m bloque del archivo que arranca en el bloque "b", se accede directamente al bloque "b+m". Su mayor desventaja es la gestión del espacio libre porque hay que poner archivos de longitud cualquiera en los bloques libres contiguos. Esta gestión se hace con first fit o best fit generalmente, y en cualquiera de los dos métodos se produce fragmentación externa (los bloques libres no alcanzan para guardar un archivo y quedan sin usarse). Otro problema es el de reservar bloques en disco previendo que el archivo crecerá, ya que no se puede saber cuánto espacio ocupara, lo que produce fragmentación interna, donde queden bloques a medio llenar o directamente queden bloques reservados sin dato alguno. En caso de que un archivo se agrande más de lo reservado, este deberá moverse a otro conjunto de bloques contiguos donde entre, y se deberá liberar el espacio antes ocupado.

Asignación enlazada:
Cada archivo es una lista enlazada de bloques de disco, pudiendo estar dichos bloque dispersos por todo el disco. Cada bloque tiene un puntero al bloque siguiente (no están disponibles al usuario). El directorio tiene un puntero al primer y último bloque de cada archivo. Para crear un archivo se crea una entrada de directorio con el puntero en NULL y el tamaño del archivo en cero. Al escribir en el archivo, el sistema de gestión de espacio libre localiza un bloque y luego de escribir los datos en el, lo enlaza al final del archivo y actualiza su tamaño. No hay fragmentación externa, ni hay que saber el tamaño máximo del archivo porque puede crecer mientras haya bloques libres. Es lento para ambos tipos de acceso por la gran cantidad de movimientos de cabezal de disco, y para hacer un acceso directo es necesario igualmente hacer un acceso secuencial para poder ubicar los bloques por lo que podemos decir que no tiene acceso directo. Otra desventaja es que cada bloque va a tendrá 4 bytes ocupados por el puntero en vez de datos útiles por lo que un archivo pesa más con este método de asignación que con el otro. Esto se puede mejorar usando clusters, ya que por ejemplo se podría tomar de a cuatro bloques como un cluster, y este solo tendría un puntero en vez de cuatro, y además los bloques serian contiguos por lo que también se reduciría la cantidad de movimientos de cabezal. Hacer esto provoca mayor fragmentación interna por ser más grandes los clusters. Otra desventaja es la fiabilidad porque si se daña un bloque, se pierde el puntero al siguiente bloque, y por lo tanto el resto del archivo.

FAT (File Allocation Table):
Es una variante de asignación enlazada porque los punteros están en una tabla indexada por número de bloque en vez de en el bloque mismo. La tabla se guarda en el principio

de cada volumen, y tiene una entrada por cada bloque de disco. El último bloque de cada archivo en la tabla tiene una marca de EOF. Los bloques libres se marcan con un valor cero, y para encontrarlos se recorre secuencialmente la tabla. Se mejora el acceso aleatorio porque se leen los punteros de la FAT en vez de cada bloque, y es conveniente tener la FAT en cache para ahorrar un acceso a disco en busca del puntero por cada acceso que se quiere hacer.

Asignación indexada:
La asignación indexada agrupa todos los punteros en una única ubicación, el bloque índice. Cada archivo tiene su propio bloque índice, que es un vector de direcciones de bloques de disco, donde la entrada n-esima del índice apunta al n-esimo bloque del archivo. El directorio contiene la dirección del bloque de índice. Cuando se crea un archivo, se pone en NULL todos los punteros del bloque de índice. Cuando se escribe en el archivo, se pide un bloque al gestor de espacio libre, y se pone su dirección en el primer puntero en NULL del bloque índice. El acceso directo es más eficiente por tener todos los punteros en un solo bloque, pero ese bloque ocupado solo por punteros por cada archivo es más espacio desperdiciado también que en enlazada. También tiene el problema de que un solo bloque de punteros no sea suficiente para archivos de gran tamaño, lo que se soluciona con los siguientes métodos:

- ✓ Esquema enlazado: se usa más de un bloque índice por archivo, y se los enlaza mediante el último puntero de cada bloque de índice.
- ✓ Índice multinivel: se usa un bloque de índice con punteros de indirección simple, o sea que cada puntero del bloque de primer nivel apunta a un bloque de segundo nivel, en el que cada puntero apunta al bloque de disco. Este método puede ampliarse hasta un tercer o cuarto nivel dependiendo el tamaño del archivo.
- ✓ Esquema combinado: usando por el UFS (UNIX File System). Consiste en tener, en este caso doce punteros directos a bloques de disco, un puntero indirecto simple, un puntero indirecto doble, y un puntero indirecto triple.

La asignación indexada tiene los mismos problemas de rendimiento que la enlazada porque los bloques de datos están dispersos por todo el disco.

Gestión del espacio libre:
Para controlar el espacio libre del disco, el sistema mantiene una lista de espacio libre. Esta lista indica todos los bloques de disco libres, o sea, que no están asignadas a ningún archivo o directorio. Para crear un archivo, se explora la lista de espacio libre hasta obtener la cantidad requerida, y se asigna el espacio al nuevo archivo, el cual se elimina de la lista de espacio libre. Cuando se borra un archivo, su espacio de disco se añade a la lista de espera libre.

Vector de bits:
Cada bloque está representado por un bit, cuyo valor es “1” si esta libre, o “0” si está ocupado. Su principal ventaja es la simplicidad y eficiencia para encontrar el primer bloque libre, ya que recorre el vector buscando un valor distinto de “0” de palabra (porque una palabra con valor “0” va a tener todos los bloques ocupados). Al encontrar la palabra distinta de “0”, se explora en busca del primer bit 1, que se corresponderá con la ubicación del primer bloque libre. El cálculo del bloque se hace como “numero de bits por palabra*palabra*numero de palabras de valor “0” + desplazamiento del primer bit de valor “1””. Son ineficientes a menos que se mantenga el vector completo en

memoria principal. Para discos de gran tamaño, el vector ocupara mucha memoria (40 Gigabytes con bloques de 1 Kilobyte representan más de 5 MB).

Lista enlazada:
Se enlazan todos los bloques de disco libres, manteniendo un puntero al primer bloque libre guardado en disco y en cache. Cada bloque tiene un puntero al siguiente. Es poco eficiente porque hay que leer cada bloque y para cada uno es una E/S en caso de querer la lista completa. Al momento de asignar un bloque se toma el primero de la lista y se actualiza el dato en disco y cache con el siguiente bloque libre.

Agrupamiento:
Es similar a la lista enlazada, pero se guardan las direcciones de “n” bloques libres en el primer bloque libre, siendo el puntero n-esimo, un puntero a otro bloque de punteros a bloques libres, y así sucesivamente. Así se pueden encontrar rápidamente un gran número de bloques libres, a diferencia del modo anterior.

Recuento:
Es similar a agrupamiento, pero en vez de tener un bloque con una lista de puteros a bloques libres, se tiene por cada entrada de la lista, el puntero a un bloque libre y el número de bloques libres contiguos a ese. Cada entrada en la lista de espacio libre estará entonces compuesta por un puntero y un contador. De este modo la lista será más corta que en agrupamiento, siempre y cuando la mayoría de los contadores sea mayor a “1” (porque el peso de cada entrada es mayor).

Capitulo 13:
Sistema de E/S

El control de los dispositivos conectados a la computadora es una de las principales focos de diseño de un sistema operativo, se necesitan diversos métodos para controlar estos dispositivos, dichos métodos forman el "Subsistema de E/S del kernel", el cual aísla el resto del kernel de la complejidad asociada al manejo de los dispositivos E/S. Para enfrentar el incremento en la variedad de dispositivos nuevos, que son muy distintos en relación a los dispositivos anteriores, se plantea una solución de hardware y software, los elementos del primero son puertos, buses y controladores, y el del segundo consiste en una estructura que haga uso de módulos específicos controladores.

Hardware de E/S:
Los dispositivos se pueden clasificar en:

- ✓ Almacenamiento (ejemplo: disco, cintas, etc.).
- ✓ Transmisión (ejemplo: tarjetas de red, módems, etc.).
- ✓ Interfaz Humana (ejemplo: pantalla, teclado, ratón, etc.).

Los dispositivos se comunican con los sistemas informáticos enviando señales a través de cables o incluso a través del aire, definiremos a continuación algunos elementos de hardware necesarios para proseguir con el tema:

Puerto: es un punto de conexión con la computadora donde se reciben las señales de los diferentes dispositivos. Este compuesto típicamente por cuatro registros:

- ✓ Registro de Estado: contiene bits que el CPU puede leer, y estos indican estados del dispositivo.
- ✓ Registro de Control: puede ser escrito por el CPU para iniciar un comando o para cambiar el modo de un dispositivo.
- ✓ Registro de Entrada de Datos: el CPU puede leer de este registro una entrada de datos.
- ✓ Registro de Salida de Datos: el CPU puede escribir en este registro una salida de datos.

Los registros de datos tienen entre uno y cuatro bytes de tamaño, únicamente que se tengan chips FIFO que permiten almacenar una pequeña ráfaga de datos hasta que el dispositivo o CPU sea capaz de recibir dichos datos.

Bus: conjunto de hilos (líneas de circuito), que representan un canal de transmisión de mensajes con un protocolo rígidamente definido que especifica que conjunto de esos mensajes se pueden enviar a través de él. Estos mensajes se transmiten mediante patrones de tensiones eléctricas aplicadas a los hilos, con unos requisitos de temporización bien definidos.

Conexión en Cascada: cuando hay varios dispositivos conectados uno seguido del otro terminando en una conexión con un puerto de computadora. Este tipo de conexiones suelen funcionar como un bus.

Hay tres tipos de buses dentro de la arquitectura de los sistemas informáticos típicos, uno es el "Bus PCI" que es el que conecta el subsistema CPU-memoria con los dispositivos de alta velocidad, y un "Bus de Expansión" que conecta los dispositivos relativamente lentos, como el teclado y los puertos serie y paralelo, y el "Bus SCSI" que conecta una serie de discos en su mayoría aunque también puede tener conectado a el otros dispositivos.

Controladora: es una serie de componentes electrónicos que permite controlar un puerto, un bus o un dispositivo. Una controladora tiene básicamente tres elementos:

- ✓ CPU.
- ✓ Memoria.
- ✓ Microcódigo.

Comunicación CPU-controladora: la controladora dispone de uno o más registros de control y de datos, en los cuales el CPU mediante una instrucción determinada de E/S, configura las líneas de bus para seleccionar el dispositivo apropiado y luego lee o escribe bits, esos bits representan patrones de comunicación. Una controladora además puede soportar un mecanismo de E/S mapeado en memoria, es decir que los registros de control están en el espacio de direcciones del CPU, por ende este puede trabajar con estos como si realizara transferencias ordinarias de lectura y escritura. Las ventajas de este tipo de esquema de comunicación es que siempre resulta más rápido que el esquema no mapeado en memoria, dado que siempre es más fácil escribir o leer múltiples bytes de la memoria que ejecutar múltiples instrucciones de E/S, pero como desventaja principal está el hecho de que el tipo de fallo más común de software consiste en modificaciones accidentales de posiciones de memoria que no les correspondían a los procesos que las modificaron, pudiendo perder información en el proceso.

Sondeo:
El protocolo de sondeo CPU-controladora es como sigue:

- ✓ El CPU lee iterativamente el bit de ocupado en el registro de estado hasta que este sea "0".
- ✓ El CPU activa el bit de escritura en el registro de comando y escribe uno byte en el registro de datos de salida.
- ✓ El CPU activa el bit de comando preparado en el registro de comando.
- ✓ Cuando la controladora se percata del bit de comando preparado, activa el bit de ocupada en el registro de estado.
- ✓ La controladora observa el bit de escritura en el registro de comando, y luego lee el registro de salida de datos para obtener el byte, y lleva a cabo la E/S hacia el dispositivo.

La controladora borra el bit de comando preparado en el registro de comando, borra el bit de error en el registro de estado, y borra el bit de ocupada en el registro de estado. Este conjunto de iteraciones se repiten para cada byte tanto si es escritura como en la secuencia anterior, como si fuese lectura (siempre que las transferencias sean de un byte). Si el dispositivo y su controladora son rápidos, este esquema resulta razonable, sin embargo si se prolonga demasiado la espera convendría que el CPU conmutara a otra tarea. El sondeo pasa a ser ineficiente cuando se ejecuta de manera repetida para

que en solo raras ocasiones esté listo el dispositivo para ser servido, en estos casos en los cuales el CPU podría conmutar a otro proceso y seguir ejecutando, se haga conveniente un esquema llamado “Interrupción”.

Interrupción:
El hardware de la CPU posee dos líneas especiales denominadas de solicitud de interrupción, que la CPU comprueba después de ejecutar cada instrucción, una de ellas es la de interrupciones no enmascarables, que se utiliza en general para sucesos tales como errores graves de hardware, y la otra es la de interrupciones enmascarables, la cual puede ser desactivada por el CPU antes de la ejecución de secuencias de instrucciones criticas que no deban ser interrumpidas, este es el tipo de línea que utilizan las controladoras de dispositivo para solicitar un servicio enviando la señal de interrupción. Cuando el CPU detecta una activación en alguna de las líneas de interrupción, este guarda el estado actual y salta a alguna de las rutinas de tratamiento de interrupciones según sea el caso (según la causa que produjo la interrupción), situada está en una dirección fija en memoria, cuando termina de ejecutar la rutina, se produce un borrado de la interrupción, una restauración de estado y un retorno para devolver al CPU al estado de ejecución anterior al que se produjera la interrupción. El mecanismo de interrupción acepta una dirección, que es un desplazamiento dentro de una tabla que contiene las direcciones de memoria fijas de las rutinas especializadas de tratamiento de interrupciones, esta tabla se denomina “Vector de Interrupciones”, el propósito de esta tabla es no tener una rutina única que tenga que analizar todas las posibles fuentes de interrupción para determinar cuál de ellas necesita servicio. En muchas ocasiones la cantidad de entradas del vector es muy pequeña en relación con la cantidad de dispositivos en el sistema, en estos casos una posible solución sería la técnica del encadenamiento de interrupciones, en la cual cada entrada del vector apunta a la cabeza de una lista de rutinas de tratamiento de interrupción, cuando se genera una interrupción se llama una a una a las rutinas de la lista correspondiente, hasta que se encuentre una rutina que pueda dar servicio a la solicitud. El mecanismo de interrupciones implementa también un sistema de niveles de prioridad de interrupción, esto le permite al CPU diferir el tratamiento de interrupciones de baja prioridad sin enmascarar todas las interrupciones, y hace posible que una interrupción de alta prioridad desaloja a otra de baja. Durante el arranque el sistema operativo comprueba los buses de hardware para determinar que dispositivos existen e instalar las rutinas de tratamiento de interrupción correspondientes, dentro del “Vector de Interrupciones”. Los sistemas operativos tienen otros usos adecuados para un mecanismo hardware y software eficiente que guarde una pequeña cantidad de información de estado del procesador y luego invoque una rutina privilegiada dentro del kernel, a modo de ejemplo se comenta la idea de cómo implementar una llamada al sistema:

- ✓ La aplicación llama a la biblioteca de llamadas al sistema.
- ✓ La llamada al sistema correspondiente comprueba los argumentos proporcionados por la aplicación, y construye una estructura de datos para entregar los argumentos al kernel.
- ✓ Finalmente la llamada al sistema ejecuta una instrucción especial denominada interrupción software con el numero de servicio del kernel deseado como operando.

Para finalizar se debe recalcar que dado que los sistemas operativos modernos utilizan extensivamente las interrupciones, se necesita que el tratamiento de las interrupciones sea eficiente para obtener un buen rendimiento del sistema.

Acceso directo a memoria:
Existe en la mayoría de las arquitecturas evitan la sobrecarga del CPU con las tareas de entrada salida programada, descargando parte de ese trabajo en un procesador de propósito especial denominado controladora de "Acceso Directo a Memoria" (DMA, Direct Memory Access). Para iniciar la transferencia DMA, el CPU describe un bloque de comando DMA en la memoria, este bloque contiene, un puntero al origen de una transferencia, un puntero al destino de la transferencia, y un contador del numero de bytes a transferir, el CPU escribe la dirección de este bloque de comandos en la controladora DMA y luego continua con sus tareas, la controladora DMA se encarga entonces de operar el bus de memoria directamente, colocando direcciones en el bus para realizar las transferencias sin ayuda de la CPU principal. El proceso de negociación entre la controladora DMA y la controladora de dispositivo, conociste en un par de líneas llamadas DMA-request y DMA-acknowledge, la controladora de dispositivo pone en la primera una señal cada vez que hay disponible una palabra de datos, esto hace que la controladora DMA tome el control del bus de memoria, como dijimos antes, coloque la dirección deseada en las líneas de dirección de memoria y coloque una señal en la línea DMA-acknowledge, cuando esto es percibido por la controladora de dispositivo, esta transfiere la palabra de datos a memoria y borra la señal DMA-request. Una vez terminada la transferencia completa, la controladora DMA interrumpe al procesador. Cuando la controladora DMA toma el control del bus de memoria impide momentáneamente al CPU acceder a la memoria principal, aunque no invalida el seguir accediendo a los elementos de datos de sus caches principal y secundaria, a este proceso se lo denomina "Robo de Ciclo". Aunque este proceso puede ralentizar los cálculos realizados por el CPU, el rendimiento global del sistema suele mejorar. En un kernel que utilice el modo protegido, el sistema operativo impide generalmente que los procesos ejecuten directamente comandos de los dispositivos, esto protege a al mismo frente a aplicaciones erróneas o maliciosas.

Interfaz de E/S de las aplicaciones:
Para implementar una interfaz que permita tratar de una forma estándar y uniforme a los dispositivos de E/S, nos debemos valer de los conceptos de ingeniería del software, como son; la abstracción, el encapsulamiento y la descomposición del software en niveles. Se deben identificar solamente un cierto conjunto de dispositivos generales, y para acceder a cada dispositivo se utilizara un conjunto estandarizado de funciones, es decir una interfaz, cada uno de estos dispositivos generales conforma un modulo de kernel denominado controlador de dispositivo (driver), y esta personalizado internamente para cada dispositivo. El propósito de la capa de controladores de dispositivo es ocultar a los ojos del subsistema de E/S del kernel las diferencias existentes entre las controladoras de dispositivo. Hacer sistemas operativos con subsistemas de E/S independiente del hardware simplifica el trabajo del desarrollador de sistemas operativos, y beneficia a los fabricantes de hardware, ya que puede desarrollar interfaces que sean compatibles con una interfaz de controladora host existente o pueden escribir controladores de dispositivo para implementar interfaz en una serie de sistemas operativos populares.

Clasificación de los dispositivos:

Flujo de caracteres o bloque: el primero transfiere los bytes uno por uno, el segundo transfiere los bytes en bloques uno por uno. Ejemplos: un teclado y un disco respectivamente.
Acceso secuencial o aleatorio: el primero transfiere los datos en un orden fijo determinado por el dispositivo, el segundo puede transferirlos de cualquiera de las ubicaciones disponibles de almacenamiento de datos. Ejemplos: una cinta y un CD-ROM respectivamente.
Síncrono o asíncrono: el primero realiza transferencias de datos con tiempos de respuesta predecibles, el segundo exhibe tiempos de respuesta irregular o no predecible. Ejemplo: cinta y teclado respectivamente.
Compartible o dedicado: el primero puede ser usado por varios procesos o hebras, el segundo no. Ejemplo: teclado y cinta respectivamente.
Velocidad de operación: varían desde unos pocos bytes por segundo a unos cuantos gigabytes por segundo. Ejemplo: comparar velocidades de diferentes dispositivos.
Lectura-escritura, solo lectura o solo escritura: el primero permite operaciones de escritura y lectura, el segundo de solo lectura, y el tercero de solo escritura. Ejemplo: disco, CR-ROM y controladora grafica respectivamente.

Dispositivos de bloques y caracteres:
Los discos son el modelo perfecto de los dispositivos de bloques, puede considerarse como su interfaz las funciones ya mencionadas en el capítulo 10, es decir Read() (para leer), Write() (para escribir), Seek() (para reposicionar el puntero de lectura-escritura). Puede ser deseable para el propio sistema operativo o aplicaciones de gestión de base de datos, acceder a dispositivos como este, como una simple matriz de bloques, otorgando un grado mayor de libertad y de responsabilidad a la aplicación si este fuese el caso, este tipo de E/S se denomina “E/S sin formato”. Otro tipo de acceso a este tipo de dispositivos es el de archivos mapeados en la memoria, que también se implementa por encima de los controladores de dispositivo, este mecanismo permite a la aplicación que lo implemente, utilizar una matriz de bytes de la memoria principal para acceder al almacenamiento en disco.
El teclado es el modelo perfecto de los dispositivos de caracteres, su interfaz consiste en las funciones como Put() (escribir un carácter), y Get() (leer un carácter).

Dispositivos de red:
Una interfaz disponible en muchos sistemas operativos, consiste en los que se dan a conocer como “sockets” de red, esta palabra en ingles hace referencia a la toma eléctrica en las paredes de una vivienda, y siguiendo con la analogía las operaciones con ellos le permiten a la aplicación:
Crear un socket (Create()).
Conectar un socket (Connect()).
Escuchar a través de un socket a la espera de una conexión remota (Listen()).
Enviar un paquete través de un socket (Send()).
Recibir un paquete través de un socket (Receive()).
También en algunos casos la interfaz de dispositivos de red podría llegar a implementar una operación denominada Select(), que permite gestionar un conjunto de “sockets”, devuelve información acerca de que “sockets” tienen un paquete a la espera de ser recibido, y que “sockets” disponen del espacio para aceptar un paquete que haya que enviar.

Relojes y temporizadores:

Las operaciones que este tipo de dispositivos permiten realizar son las siguientes:

- ✓ Obtener la hora actual.
- ✓ Obtener el tiempo transcurrido.
- ✓ Obtener la provocación de la ejecución de alguna operación en un instante "t".

No existe estándar alguno para la interfaz que cada sistema operativo distinto le provee a la aplicación para utilizar alguna de estas tres operaciones. El hardware necesario para realizar para la segunda y la tercera operaciones se denomina "Temporizador de Intervalo Programable", se puede configurar este para esperar una cierta cantidad de tiempo y luego generar una interrupción. Este mecanismo lo utiliza el planificador (aquellos que implementan cierto tipo de algoritmos de planificación) para generar una interrupción que provoque el desalojo de un proceso al final de su correspondiente franja temporal de ejecución. También es utilizado tanto por el subsistema de E/S para volcar en el disco los búferes cache sucios, como por el subsistema de red para cancelar operaciones que por fallos de red o congestión estén progresando de forma muy lenta. La interfaz para los procesos, consiste en relojes virtuales, implementados mediante una lista de interrupciones deseadas por los procesos y por las rutinas del kernel, ordenada del tiempo más próximo al tiempo más lejano. Mediante esta lista el kernel carga al temporizador de a un elemento de la lista por vez, por cada una de esas veces, el temporizador interrumpe, y el kernel señaliza dicho suceso al proceso correspondiente al elemento de la lista, cuando termina, vuelve a repetir la operación.

E/S Bloqueante y no bloqueante:
A continuación enunciamos los conceptos de estos dos tipos de mecanismos de E/S:

- ✓ Bloqueante: cuando una aplicación invoca un tipo de llamada al sistema bloqueante, se suspende la ejecución de la aplicación, en ese instante la PCB de la aplicación pasa de la cola de preparados a la cola correspondiente al dispositivo con el cual pretende operar, y se le modifica su estado de "Listo" a "Esperando", luego de finalizar la operación se realizara el camino inverso al anterior, y se le entregaran a la aplicación los valores devueltos por la llamada al sistema.
- ✓ No bloqueante: cuando una aplicación invoca un tipo de llamada al sistema no bloqueante, no se detiene la ejecución de la aplicación durante un periodo prolongado de tiempo, en lugar de ellos la llamada vuelve rápidamente, y se le entregan a la aplicación los valores disponibles hasta el momento devueltos por la llamada al sistema.

Una alternativa a este ultimo tipo de llamadas al sistemas son las llamadas asíncronas, en las cuales se vuelve inmediatamente sin espera a que la operación de E/S se complete, lo que le permite a la aplicación continuar ejecutando su código, cuando en cualquier instante en el futuro se complete la operación, se le comunicara a la aplicación mediante alguna variable de la aplicación, señal, interrupción de software o ejecutando una rutina de retro llamada fuera del flujo de la aplicación.

Subsistema de E/S del kernel:
Muchos de los servicios relacionados con E/S que provee el kernel son proporcionados por el subsistema de E/S, el cual utiliza como basamento el hardware y los controladores de dispositivo. También es responsable de protegerse a sí mismo de los procesos erróneos y usuarios maliciosos. A continuación mencionaremos algunos de los servicios antes mencionados:

- ✓ Planificación de E/S:
  Implica ordenar un conjunto de solicitudes de E/S según un criterio adecuado, a fines de mejorar el rendimiento global del sistema, compartir el acceso a los dispositivos equitativamente entre los procesos y reducir el tiempo medio de espera requerido para que la E/S se complete. Cuando un kernel implementa el mecanismo de E/S asíncrona, debe ser capaz de controlar múltiples solicitudes de E/S al mismo tiempo, para realizar esta tarea el sistema operativo puede asociar a la cola de espera una "Tabla de Estado del Dispositivo", con una entrada por cada dispositivo. Cada una de dichas entradas contendrá el tipo, la dirección y el estado del dispositivo. Otras formas de mejorar la eficiencia de la computadora, es utilizando espacio de almacenamiento en la memoria principal o en el disco mediante técnicas de almacenamiento en búfer, almacenamiento en cache y gestión de colas.
- ✓ Almacenamiento en búfer:
  Un búfer es un área de memoria que almacena datos mientras se están transfiriendo entre dos dispositivos o entre un dispositivo y una aplicación. Una de las razones de este mecanismo es amortiguar las velocidades de transferencia de datos entre el productor y el consumidor en un flujo de datos. Existe la convencional técnica de un único búfer donde el productor va llenando el búfer, y una vez lleno, el consumidor vacía todo el búfer, pero también se pudrían utilizar dos búferes con un esquema donde el productor llene el primer búfer, y luego empiece a llenar el segundo búfer, mientras el consumidor va vaciando el primero, entonces para cuando el productor halla llenado el segundo búfer, ya el consumidor habrá vaciado el primero, lo que le permitirá al productor conmutar nuevamente al primer búfer mientras el consumidor vacía el segundo búfer, y así sucesivamente. La idea de la antes mencionada técnica llamada de "Doble Búfer", es que el productor no se quede sin producir en espera de que el consumidor consuma el búfer que se acaba de llenar. Otras dos razones de la utilización de búferes en el sistema, consisten en primer lugar, en amortiguar las diferencias de tamaño de transferencia de datos entre el productor y el consumidor, y en segundo lugar soportar semántica de copia para garantizar que la versión de datos tomada por el consumidor, sea la que genero el productor.
- ✓ Almacenamiento en cache:
  Es una región de memoria rápida que alberga copias de ciertos datos, dado que el acceso a esas copias es más rápido que el acceso a sus originales. La diferencia entre una cache y un búfer es que el último puede almacenar la única copia existente de los datos.
- ✓ Gestión de colas y reserva de dispositivos:

Una cola de dispositivo es un búfer que almacena la salida dirigida a un dispositivo, hasta que se pueda realizar dicha operación. El sistema operativo provee una interfaz de control que permite a usuarios y administradores del sistema visualizar la cola de solicitudes, eliminar trabajos no deseados, suspender la operación mientras se solventa algún error de dispositivo, etc.

- ✓ Tratamiento de Errores:
  Los dispositivos y las transferencias de E/S pueden fallar de muchas formas, debido a razones transitorias (Ejemplo: sobrecarga de una red), o razones permanentes (Ejemplo: falla una controladora de disco). A menudo los sistemas operativos pueden compensar los de la primera clase, pero en el caso de los permanentes muy difícilmente el sistema operativo pueda recuperarse. Un sistema operativo que utilice memoria protegida puede defenderse frente a muchos tipos de errores de hardware y de aplicaciones, de modo que cada pequeño error no provoque un fallo completo del sistema. Como regla general una llamada E/S siempre retorna un bit de información mediante el que se indicara si la llamada ha tenido éxito o no.
- ✓ Protección de E/S:
  Los mecanismos de protección en contra de procesos de usuarios que accidental o deliberadamente intentan interrumpir la operación normal del sistema son los siguientes:
    - Todas las instrucciones de E/S son privilegiadas: los usuarios no pueden ejecutar instrucciones de E/S directamente, sino que tienen que hacerlo a través del sistema operativo.
    - Uso del sistema de protección de memoria: se protegen todas las ubicaciones de memoria mapeada y de los puertos E/S frente a accesos de los usuarios.
- ✓ Estructuras de datos del kernel:
  El kernel necesita mantener información acerca del uso de los componentes de E/S, y lo implementa mediante diversas estructuras de datos internas (Ejemplo: Tabla Global de Archivos Abiertos).

Rendimiento:
La E/S es uno de los factores que más afectan al rendimiento del sistema. Podemos emplear diversos principios para mejorar la eficiencia de la E/S, a continuación mencionamos algunos:

- ✓ Reducir el número de cambios de contexto.
- ✓ Reducir el número de veces que los datos deben copiarse en memoria mientras pasa del dispositivo a la aplicación o viceversa.
- ✓ Reducir la frecuencia de las interrupciones utilizando transferencias de gran tamaño, controladoras inteligentes y mecanismos de sondeo (con baja espera activa).
- ✓ Incrementar la concurrencia utilizando controladoras preparadas para DMA o canales, con objeto de descargar a la CPU de las operaciones simples de copia de datos.
- ✓ Desplazar las primitivas de procesamiento al hardware, para permitir que se ejecuten en las controladoras de dispositivo, de forma concurrente con las operaciones de CPU y bus.
- ✓ Equilibrar el rendimiento de la CPU, del subsistema de memoria, del bus y de la E/S, para no generar tiempos muertos por sobrecarga.