

Un sistema operativo (SO) explota los recursos de hardware de uno o más procesadores para ofrecer un conjunto de servicios a los usuarios del sistema. El SO también gestiona la memoria secundaria y los dispositivos de entrada/salida (E/S) en nombre de los usuarios.

Se le puede dar una perspectiva en dos sentidos:

- I. **Perspectiva de arriba hacia abajo:** se le realiza una abstracción con respecto a la arquitectura, que es el conjunto de instrucciones, organización de memoria, E/S y la estructura del bus. El SO oculta el hardware y presenta los programas, siendo estas abstracciones más simples de manejar. Los programas de aplicación son los clientes, para llamarlo de alguna manera, del SO. Esto nos permite hacer una comparación con el uso de escritorio y el uso de comandos de texto, dándonos comodidad y amigabilidad con la computadora.
- II. **Perspectiva de abajo hacia arriba:** se ve al SO como un administrador de recurso, este lleva a cabo la administración de los recursos del hardware de uno o más procesadores; provee un conjunto de servicios a los usuarios del sistema, maneja la memoria secundaria y dispositivos de E/S, ejecuta simultáneamente programas y da multiplexación en tiempo (CPU) y en espacio (memoria).

Elementos básicos de una computadora: en un alto nivel, un sistema informático consta de procesador, memoria y componentes de E/S, con uno o más módulos de cada tipo. Estos componentes están interconectados de alguna forma para llevar a cabo la función principal del computador, que es ejecutar programas. Así pues, se tienen cuatro elementos principales:

- **Procesador:** controla la operación del computador y lleva a cabo las funciones de procesamiento de datos, cuando hay un solo procesador se denomina CPU.
- **Memoria principal:** almacena los datos y programas, está es normalmente volátil; también se la conoce como memoria primaria o real.
- **Componentes de E/S:** llevan a cabo la relación entre el usuario del mundo exterior y la computadora, pueden ser los dispositivos de memoria secundaria, equipamiento de comunicación, monitor, teclado o mouse.
- **Interconexión de sistemas:** ciertos mecanismos y estructuras que permiten la comunicación entre procesadores, memoria principal y los módulos de E/S.

Estos 4 elementos fundamentales se consideran componentes de alto nivel. El procesador es quien normalmente lleva el control, una de sus funciones es intercambiar datos con la memoria, para este propósito, hace uso de dos registros internos a él MAR (registro de dirección de memoria), el cuál especifica la dirección en memoria de la próxima lectura o escritura, y MBR (registro de datos), el cual contiene los datos que van a ser escritos a memoria o que fueron leídos por la misma. De manera similar, un registro de direcciones de E/S (IOAR, Input Output Address Register) especifica un dispositivo particular de E/S. Un registro intermedio de E/S (IOBR, Input Output Buffer Register) se utiliza para intercambiar datos entre un módulo de E/S y el procesador. Un módulo de memoria consta de un conjunto de ubicaciones definidas por direcciones enumeradas secuencialmente.

Dentro del procesador, hay un conjunto de registros que ofrecen un nivel de memoria que es más rápido y pequeño que la memoria principal. Los registros del procesador sirven para dos funciones:

- I. **Registros visibles por el usuario:** un programador de lenguaje de máquina o ensamblador puede minimizar las referencias a memoria principal mediante un uso óptimo de estos

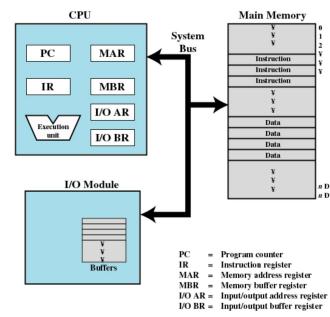


Figure 1.1 Computer Components: Top-Level View

PC = Program counter
 IR = Instruction register
 MAR = Memory address register
 MBR = Memory buffer register
 IO AR = Input/output address register
 IO BR = Input/output buffer register

registros. Con lenguajes de alto nivel, un compilador que optimice código intentara hacer una selección inteligente de que variables asignar a registros y cuales a ubicaciones de la memoria principal.

Un registro visible de usuario es aquel que puede ser referenciado por medio del lenguaje de máquina que ejecuta el procesador y es, por lo general, accesible para todos los programas, incluyendo tanto los programas de aplicación como los del sistema. Las clases de registro que, normalmente, están disponibles, son los registros de datos, los registros de dirección y los registros de códigos de condición. Estos registros se pueden dividir en dos clases:

- **Registros de datos:** pueden ser asignados por programador a diversas funciones.
- **Registros de dirección:** contiene direcciones en la memoria principal de datos e instrucciones o una parte de la dirección que se utiliza en el cálculo de la dirección completa, estos pueden ser:
 - **Index (registro de índice):** sirve para el direccionamiento indexado.
 - **Segment pointer (puntero de segmento):** con direccionamiento segmentado, la memoria se divide en segmentos que son bloques de palabras de tamaño variable, una referencia a memoria consta de una referencia a un segmento particular y un desplazamiento dentro del segmento.
 - **Stack pointer (puntero de pila):** si hay direccionamiento de pilas visibles para los usuarios, la pila estará en la memoria principal, existiendo un registro dedicado a señalar la cabeza de la misma.
- II. **Registros de control y estado:** son utilizados por el procesador para el control de las operaciones o por rutinas privilegiadas del sistema operativo para controlar la ejecución de los programas.
Varios registros se emplean para controlar las operaciones del procesador. En la mayoría de las maquinas, la mayor parte de estos registros no son visibles para los usuarios. Algunos de ellos pueden estar accesibles a las instrucciones de maquina ejecutadas en un modo de control o modo del sistema. Además de los registros MAR, MBR, IOAR e IOBR, otros registros esenciales en la ejecución de instrucciones son:
 - **Program counter (PC):** contiene la dirección de la instrucción a ser leída.
 - **Instruction register (IR):** contiene la última instrucción leída.
 - **Program status Word (PSW):** contiene la información de estado, normalmente, contiene códigos de condición junto a otra información de estado, este habilita/deshabilita interrupciones e indica el modo de ejecución (Supervisor/user).

Ejecución de instrucciones: la tarea básica que realiza un computador es la ejecución de los programas. El programa a ejecutar consta de un conjunto de instrucciones almacenadas en memoria, el procesador lleva a cabo el trabajo, ejecutando las instrucciones especificadas en el programa. La ejecución de un programa consiste en la repetición de este proceso de lectura y ejecución de la instrucción, la ejecución de la instrucción puede involucrar varias operaciones y depende de la naturaleza de la instrucción.

El procesamiento requerido para una instrucción simple se llama ciclo de instrucción, este cuenta con dos pasos, ciclo de lectura (fetch) y ciclo de ejecución. La ejecución del programa se detiene solo si se apaga la máquina, ocurre algún tipo de error irrecuperable o se encuentra una instrucción en el programa que detiene el computador.

Al comienzo de cada ciclo de instrucción, el procesador lee una instrucción de la memoria. En un procesador típico habrá un registro PC, que se usa para llevar la cuenta de cuál es la próxima instrucción a leer. A menos que se diga otra cosa, el procesador siempre incrementara el PC después de leer cada instrucción, de forma que después se lea la instrucción siguiente en la

secuencia. La instrucción leída se carga en un registro del procesador conocido como registro de instrucción (IR), la cual especifica cual es la acción que el procesador llevará a cabo; el procesador interpreta la instrucción y realiza la acción requerida. En general, estas acciones pueden clasificarse en las siguientes cuatro categorías:

- I. **Procesador-memoria:** se transfieren datos del procesador a memoria o viceversa.
- II. **Procesador-E/S:** se transfieren datos desde o hacia un dispositivo periférico, realizándose la transferencia entre el procesador y un módulo de E/S.
- III. **Procesamiento de datos:** el procesador realiza alguna operación aritmética o lógica sobre los datos.
- IV. **Control:** la instrucción pide que se altere la secuencia de ejecución.

Interrupciones: casi todos los computadores tienen un mecanismo mediante el cual otros módulos de E/S pueden interrumpir la ejecución del procesador, estas aparecen como una vía para mejorar la eficiencia del procesamiento, ya que no todos presentan la misma velocidad que el procesador. Las clases de interrupciones que pueden aparecer son:

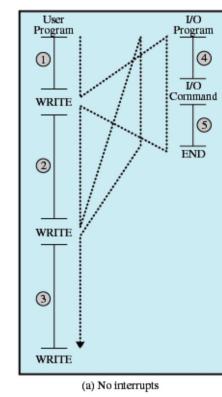
- I. **Programa:** generadas por alguna condición que se produce como resultado de la ejecución de una instrucción como el desbordamiento aritmético, la división por cero, el intento de ejecutar una instrucción ilegal de la máquina o una referencia a una zona de memoria fuera del espacio permitido por el usuario.
- II. **Reloj:** generadas por un reloj interno del procesador, esto permite al SO llevar a cabo ciertas funciones con determinada regularidad.
- III. **E/S:** generadas por controladores de E/S, para indicar que una operación ha terminado normalmente o para indicar diversas condiciones de error.
- IV. **Fallo del hardware:** generadas por fallos tales como un corte de energía o un error de paridad de la memoria.

En la figura el programa de usuario lleva a cabo una serie de llamadas a ESCRIBIR, intercaladas con el procesamiento. Los segmentos de código 1, 2 y 3 se refieren a secuencias de instrucciones que no implican E/S. Las llamadas a ESCRIBIR son, en realidad, llamadas a un programa de E/S, que es una utilidad del sistema que llevará a cabo la operación concreta de E/S. El programa de E/S consta de tres secciones:

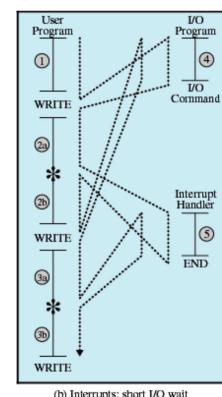
- Una secuencia de instrucciones, etiquetada con un 4 en la figura, de preparación para la operación concreta de E/S. Esto puede incluir la copia de los datos de salida hacia un buffer especial y preparar los parámetros de la orden del dispositivo.
- La orden concreta de E/S. Sin el uso de interrupciones, una vez que se emita esta orden, el programa debe esperar a que el dispositivo de E/S lleve a cabo la función pedida, el programa puede esperar simplemente ejecutando de forma repetida una operación que compruebe si ya se realizó la E/S.
- Una secuencia de instrucciones, etiquetada con Un 5 en La figura, para completar la operación. Esto puede incluir la activación de un código de condición que indique el éxito o el fracaso de la operación.

Debido a que la operación de E/S puede tardar un tiempo relativamente grande en terminar, el programa de E/S puede quedar colgado esperando a que se complete la operación; así pues, el programa de usuario se detendrá por un tiempo considerable en el momento de la llamada a ESCRIBIR.

Con las interrupciones, el procesador se puede dedicar a la ejecución de otras instrucciones mientras una operación de E/S está en proceso, dicha operación de E/S se lleva a cabo concurrentemente con la ejecución de las instrucciones



(a) No interrupts



(b) Interrupts; short I/O wait

del programa de usuario; cuando el dispositivo de E/S esté disponible, es decir, cuando esté preparado para aceptar más datos desde el procesador, el módulo de E/S de dicho dispositivo enviará una señal de solicitud de interrupción al procesador, el procesador responde suspendiendo la operación del programa en curso y saltando a un programa que da servicio al dispositivo de E/S en particular, conocido como rutina de tratamiento de la interrupción (interrupt handler), reanudando la ejecución original después de haber atendido al dispositivo.

Para dar cabida a las interrupciones, se añade un ciclo de interrupción al ciclo de Instrucción, en el ciclo de interrupción, el procesador comprueba si ha ocurrido alguna interrupción, lo que se indicará con la presencia de una señal de interrupción, si no hay interrupciones pendientes, el procesador sigue con el ciclo de lectura y trae la próxima instrucción del programa en curso; si hay una interrupción pendiente, el procesador suspende la ejecución del programa en curso y ejecuta una rutina de tratamiento de la interrupción. La rutina de tratamiento de la interrupción forma parte generalmente del sistema operativo, normalmente este programa determina la naturaleza de la interrupción y realiza cuantas acciones sean necesarias, de hecho, en el ejemplo que se ha estado siguiendo, la rutina de tratamiento determina el módulo de E/S que generó la interrupción y puede saltar a un programa que escribirá más datos a dicho modulo, cuando termina la rutina de tratamiento de la interrupción, el procesador puede reanudar la ejecución del programa de usuario en el punto en que sucedió la interrupción.

El acontecimiento de una interrupción desencadena una serie de sucesos, tanto en el hardware del procesador como en el software; cuando un dispositivo de E/S completa una operación se produce:

I. En el hardware:

- El controlador del dispositivo u otro sistema hardware genera una interrupción.
- El procesador finaliza la ejecución de la instrucción en curso.
- El procesador recibe el recibo de la interrupción.
- El procesador inserta la PWS y el PC en la pila.
- El procesador carga el nuevo valor del PC dependiendo de la interrupción.

II. En el software:

- Salvar el resto de la información de estado del proceso.
- Interrupción del proceso.
- Restaurar la información de estado del proceso.
- Restaurar los valores anteriores de PSW y PC.

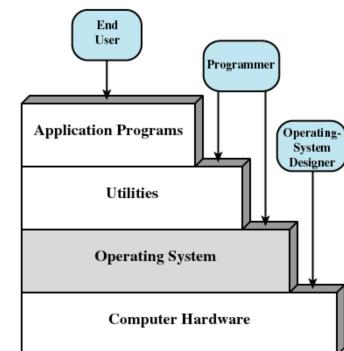
Esto se implementa para una sola interrupción, se pueden producir múltiples interrupciones. Hay dos enfoques para tratar las interrupciones múltiples. El primero es inhabilitar las interrupciones mientras se esté procesando una. Una *interrupción inhabilitada* quiere decir que el procesador ignorara la señal de interrupción, si durante este tiempo se produce una interrupción, esta generalmente quedará pendiente y será comprobada por el procesador después que este habilite las interrupciones. De este modo, cuando un programa de usuario está ejecutándose y se produce una interrupción, se inhabilitan inmediatamente las interrupciones, después de terminar la rutina que trata la interrupción, se habilitan las interrupciones antes de reanudar el programa de usuario y el procesador comprueba si se ha producido alguna interrupción adicional. Este enfoque es simple y elegante porque las interrupciones se tratan en un orden estrictamente secuencial. La limitación de este enfoque es que no tiene en cuenta las prioridades relativas o necesidades críticas en tiempo.

Un segundo enfoque es definir prioridades para las interrupciones y permitir que una interrupción de una prioridad más alta pueda interrumpir a la rutina de tratamiento de una interrupción de prioridad más baja.

Funciones y objetivos de un SO: un sistema operativo es un programa que controla la ejecución de los programas de aplicación y que actúa como interfaz entre el usuario de un computador y el hardware de la misma. Puede considerarse que un sistema operativo tiene tres objetivos o lleva a cabo tres funciones:

- I. **Comodidad:** hace que un computador sea más cómoda de utilizar.
- II. **Eficiencia:** permite que los recursos de un sistema informático se aprovechen de manera más eficiente.
- III. **Capacidad de evolución:** debe construirse de modo que permita el desarrollo efectivo, la verificación y la introducción de nuevas funciones en el sistema y, a la vez, no interferir en los servicios que brinda.

SO como interfaz usuario computadora: el hardware y el software que se utilizan para proveer de aplicaciones a los usuarios pueden contemplarse de forma estratificada o jerárquica, al usuario de estas aplicaciones se le llama usuario final y, generalmente, no tiene que ocuparse de la arquitectura del computador. Por tanto, el usuario final ve al sistema informático en términos de aplicaciones. Las aplicaciones pueden construirse con un lenguaje de programación y son desarrolladas por programadores de aplicaciones. Para facilitar esta tarea, se ofrecen una serie de programas de sistemas que se denominan utilidades e implementan funciones muy utilizadas que ayudan a la creación de los programas, la gestión de los archivos y el control de los dispositivos de E/S, los programadores hacen uso de estos servicios en el desarrollo de una aplicación y esta, mientras se está ejecutando, invoca a estas utilidades para llevar a cabo ciertas acciones, el programa de sistemas más importante es el sistema operativo, que oculta al programador los detalles del hardware y le proporciona una interfaz cómoda para utilizar el sistema, es decir, actúa como mediador, facilitándole al programador y a los programas de aplicación el acceso y uso de todas esas características y servicios. Algunos de los servicios que ofrece el sistema operativo son:



- **Creación de programas:** ofrece una variedad de características y servicios, tales como los editores y los depuradores (debuggers), para ayudar al programador en la creación de programas, normalmente, estos servicios están en forma de programas de utilidad que no forman realmente parte del sistema operativo, pero que son accesibles a través del mismo.
- **Ejecución de programas:** para ejecutar un programa se necesita un cierto número de tareas, las instrucciones y los datos se deben cargar en la memoria principal, los archivos y los dispositivos de E/S se deben inicializar y se deben preparar otros recursos. El SO administra todas estas tareas para el usuario.
- **Accesos a los dispositivos de E/S:** cada dispositivo de E/S requiere un conjunto propio y peculiar de instrucciones o de señales de control para su funcionamiento, donde el SO tiene en cuenta estos detalles de modo que el programador pueda pensar en forma de lecturas y escrituras simples.
- **Acceso controlado de los archivos:** en el caso de los archivos, el control debe incluir una comprensión, no solo de la naturaleza del dispositivo de E/S (controlador de disco, controlador de cinta) sino del formato de los archivos y del medio de almacenamiento, una vez más, es el SO el que se encarga de los detalles, es mas, en el caso de sistemas con varios usuarios trabajando simultáneamente, es el sistema operativo el que brinda los mecanismos de control para controlar el acceso a los archivos.
- **Acceso al sistema:** en el caso de un sistema compartido o público, el sistema operativo controla el acceso al sistema como un todo y a los recursos específicos del sistema, las

funciones de acceso pueden brindar protección, a los recursos y a los datos, ante usuarios no autorizados y debe resolver los conflictos en la propiedad de los recursos.

- **Detección y respuesta a errores:** cuando un sistema informático está en funcionamiento pueden producirse varios errores, entre estos se incluyen los errores internos y externos del hardware, tales como los errores de memoria, fallos o mal funcionamiento de dispositivos y distintos tipos de errores de software, como el desbordamiento aritmético, el intento de acceder a una posición prohibida de memoria y la incapacidad del sistema operativo para satisfacer la solicitud de una aplicación, en cada caso, el sistema operativo debe dar una respuesta que elimine la condición de error con el menor impacto posible sobre las aplicaciones que están en ejecución, la respuesta puede ser desde terminar el programa que produjo el error, hasta reintentar la operación o, simplemente, informar del error a la aplicación.
- **Contabilidad:** un buen sistema operativo debe recoger estadísticas de utilización de los diversos recursos y supervisar los parámetros de rendimiento tales como el tiempo de respuesta, para cualquier sistema, esta información es útil para anticiparse a la necesidad de mejoras futuras y para ajustar el sistema y así mejorar su rendimiento, en un sistema multiusuario, la información puede ser utilizada con propósito de cargar en cuenta.

El SO como administrador de recursos: un computador es un conjunto de recursos para el traslado, almacenamiento y proceso de datos y para el control de estas funciones, el SO es el responsable de la gestión de estos recursos. Administrando los recursos del computador, el SO tiene el control sobre las funciones básicas de la misma. Pero este control se ejerce de una manera curiosa, normalmente, se piensa en un mecanismo de control como algo externo a lo controlado o, al menos, como algo distinto y una parte separada de lo controlado. Este no es el caso de un sistema operativo, que no es habitual como mecanismo de control en dos aspectos:

- El SO funciona de la misma manera que el software normal de un computador, es decir, es un programa ejecutado por el procesador.
- El SO abandona con frecuencia el control y debe depender del procesador para recuperarlo.

El SO es, de hecho, nada más que un programa del computador, como otros programas de computador, da instrucciones al procesador. La diferencia clave está en el propósito del programa, el SO dirige al procesador en el empleo de otros recursos del sistema y en el control del tiempo de ejecución de otros programas. Pero para que el procesador pueda hacer estas cosas, debe cesar la ejecución del programa del sistema operativo y ejecutar otros programas, así pues, el sistema operativo cede el control al procesador para hacer algún trabajo "útil" y luego lo retoma durante el tiempo suficiente para preparar el procesador para llevar a cabo la siguiente parte del trabajo. Una parte del SO está en la memoria principal, en esta parte está el núcleo (Kernel), que incluye las funciones utilizadas con más frecuencia en el sistema operativo y, en un momento dado, puede incluir otras partes del SO que estén en uso. El resto de la memoria principal contiene datos y otros programas de usuario, el sistema operativo decide cuándo puede utilizarse un dispositivo de E/S por parte de un programa en ejecución y controla el acceso y la utilización de los archivos. El procesador es, en sí mismo, un recurso y es el SO el que debe determinar cuánto tiempo del procesador debe dedicarse a la ejecución de un programa de usuario en particular, en el caso de sistemas multiprocesador, la decisión debe distribuirse entre todos los procesadores.

Evolución de un SO: un sistema operativo importante evolucionara en el tiempo por una serie de razones:

- **Soportar nuevos tipos de hardware:** las versiones más recientes se han modificado para aprovechar las capacidades de paginación, además, el empleo de terminales gráficos y

terminales de pantalla completa, en lugar de los terminales de líneas, pueden influir en el diseño de los sistemas operativos.

- **Nuevos servicios:** como respuesta a Las demandas del usuario o a las necesidades de los administradores del sistema, el sistema operativo ampliará su oferta de servicios.
- **Correcciones:** desafortunadamente, el sistema operativo tiene fallos que se descubrirán con el curso del tiempo y que es necesario corregir, por supuesto, estas correcciones pueden introducir nuevos fallos a su vez y así sucesivamente.

Evolución de los SO: para comprender los requisitos básicos de un sistema operativo y el significado de las características principales de un SO contemporáneo, resulta útil considerar como han evolucionado los sistemas operativos a lo largo de los años:

- **Proceso en serie:** en los primeros computadores, de finales de los 40 hasta mediados de los 50, el programador interactuaba directamente con el hardware; no había SO, la operación con estas máquinas se efectuaba desde una consola consistente en unos indicadores luminosos, unos conmutadores, algún tipo de dispositivo de entrada y una impresora. Los programas en código máquina se cargaban a través del dispositivo de entrada, si se detiene el programa por un error, la condición de error se indicaba mediante los indicadores luminosos, el programador podía examinar los registros y la memoria principal para determinar la causa del error, si el programa continuaba hasta su culminación normal, la salida aparecería en la impresora. Los problemas que se generaban eran dos principalmente:
 - I. **Planificación:** la mayoría de las instalaciones empleaban un formulario de reserva de tiempo de máquina, normalmente, un usuario podía reservar bloques de tiempo en múltiplos de media hora o algo por el estilo. Un usuario podía reservar una hora y terminar a los 45 minutos; esto daba como resultado un desperdicio del tiempo del computador, por el contrario, el usuario podía tener dificultades, no terminar en el tiempo asignado y verse forzado a parar sin haber solucionado el problema.
 - II. **Tiempo de preparación:** un programa sencillo, llamado trabajo, cargaba un compilador y un programa en lenguaje de alto nivel (programa fuente) en la memoria, salvaba el programa compilado (programa objeto) y luego montaba y cargaba el programa objeto junto con las funciones comunes. Cada uno de estos pasos podía implicar montar y desmontar cintas o preparar paquetes de tarjetas, si se producía un error, el infortunado usuario tenía que volver al inicio de este proceso de preparación. De este modo, se perdía un tiempo considerable en preparar un programa para su ejecución.
- **Sistemas sencillos de procesos por lotes (batch):** las primeras máquinas eran muy caras y, por tanto, era importante maximizar la utilización de las mismas. El tiempo desperdiciado por la planificación y la preparación era inaceptable, para mejorar el uso, se desarrolló el concepto de sistema operativo por lotes (batch). El primer sistema operativo por lotes fue desarrollado a mediados de los 50 por la General Motors para usar en un IBM 701 [WEI81].
La idea central que esta detrás del esquema sencillo de proceso por lotes es el uso de un elemento de software conocido como monitor. Con el uso de esta clase de SO, los usuarios ya no tenían acceso directo a la máquina, en su lugar, el usuario debía entregar los trabajos en tarjetas o en cinta al operador del computador, quien agrupaba secuencialmente los trabajos por lotes y ubicaba los lotes enteros en un dispositivo de entrada para su empleo por parte del monitor, cada programa se construía de modo tal

que volviera al monitor al terminar su procesamiento y, en ese momento, el monitor comenzaba a cargar automáticamente el siguiente programa.

Desde el punto de vista del monitor, él es quien controla la secuencia de sucesos, para que esto sea posible, gran parte del monitor debe estar siempre en memoria principal y disponible para su ejecución. El monitor lee los trabajos uno a uno del dispositivo de entrada, a medida que lo lee, el trabajo actual se ubica en la zona del programa de usuario y el control pasa al trabajo, cuando el trabajo termina, se devuelve el control al monitor, quien lee inmediatamente un nuevo trabajo. Los resultados de cada trabajo se imprimen y entregan al usuario.

Desde el punto de vista del procesador, en un cierto momento, el procesador estará ejecutando instrucciones de la zona de memoria principal que contiene al monitor, estas instrucciones hacen que el trabajo siguiente sea leído en otra zona de la memoria principal, una vez que el trabajo se ha leído, el procesador encuentra en el monitor una instrucción de desvió que ordena al procesador continuar la ejecución en el inicio del programa de usuario, el procesador ejecuta entonces las instrucciones del programa de usuario hasta que encuentre una condición de finalización o de error.

De este modo, la frase "el control se le pasa al trabajo" quiere decir simplemente que el procesador pasa a leer y ejecutar instrucciones del programa de usuario, mientras que la frase "el control vuelve al monitor" quiere decir que el procesador pasa ahora a leer y ejecutar las instrucciones del programa monitor, es el monitor el que gestiona el problema de la planificación, se pone en cola un lote de trabajos y estos son ejecutados tan rápido como es posible, sin que haya tiempo alguno de desocupación. Este tipo de trabajo se denomina perforación de tarjetas, punching cards, el monitor lee una tarjeta y la carga al compilador adecuado desde el dispositivo de almacenamiento masivo, este traduce el programa usuario en código objeto. Durante la ejecución del programa de usuario, cada instrucción de entrada origina la lectura de una tarjeta de datos, la instrucción de entrada en el programa del usuario hace que se invoque una rutina de entrada, que forma parte del sistema operativo, la rutina de entrada se asegura de que el programa de usuario no ha leído accidentalmente una tarjeta JCL (Job Control Lenguaje), si esto sucede, se produce un error y el control se transfiere al monitor. Al terminar un trabajo, con o sin éxito, el monitor recorre las tarjetas de entrada hasta encontrar la próxima tarjeta JCL. De este modo, el sistema se protege contra un programa que tenga tarjetas de datos de más o de menos.

Ciertas instrucciones son designadas como privilegiadas y pueden ser ejecutadas solo por el monitor. Si el procesador encuentra una instrucción tal, cuando está ejecutando el programa del usuario, se producirá una interrupción de error, entre las instrucciones privilegiadas se encuentran las instrucciones de E/S, de forma que el monitor retenga el control de todos los dispositivos de E/S. Esto impide, por ejemplo, que un programa de usuario lea accidentalmente instrucciones de control que son del trabajo siguiente, si un programa de usuario desea realizar una E/S, debe solicitarse al monitor que haga la operación por él, si el procesador encuentra una instrucción privilegiada cuando está ejecutando un programa de usuario, el hardware del procesador la considera como un error y transfiere el control al monitor.

- **Sistemas por lotes con multiprogramación:** aun con el secuenciamiento automático de los trabajos ofrecido por un SO sencillo por lotes, el procesador está desocupado a menudo, el problema es que los dispositivos de E/S son lentos comparados con el procesador, el procesador gasta parte del tiempo ejecutando hasta que encuentra una instrucción de E/S. Entonces debe esperar a que concluya la instrucción de E/S antes de continuar. Esta

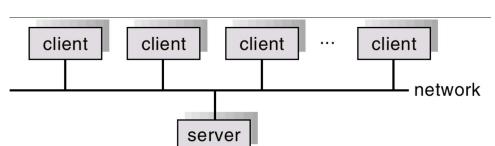
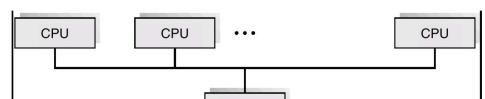
ineficiencia no es necesaria. Se sabe que hay memoria suficiente para almacenar el sistema operativo (el monitor residente) y un programa de usuario. Supóngase que hay espacio suficiente para el sistema operativo y dos programas usuarios, ahora, cuando un trabajo necesite esperar una E/S, el procesador puede cambiar al otro trabajo, que probablemente no estará esperando a la E/S, además, se podría ampliar la memoria para almacenar tres, cuatro o más programas y conmutar entre todos ellos, este proceso es conocido como multi-programador o multitarea. Este es el punto central de los SO modernos. Con este la operación de los sistemas batch se vio beneficiada del spooling de las tareas, al solapar la E/S de una tarea de la ejecución de otra, al estar las tareas cargadas en el disco, ya no era necesario ejecutarlas en el orden que fueron cargadas, ya que el SO mantiene varias tareas en memoria al mismo tiempo. La secuencia de programas es de acuerdo a la prioridad u orden de llegada, cuando el proceso necesita realizar una operación de E/S, la CPU en lugar de permanecer ociosa es utilizada para otro proceso, luego de haberse completado al atención a la interrupción, el control puede o no retornar al programa que se estaba ejecutando al momento de interrupción.

- **Sistemas de tiempo compartido:** con el uso de la multiprogramación, el tratamiento por lotes puede llegar a ser bastante eficiente, sin embargo, para muchas tareas, es conveniente suministrar un modo en que el usuario interactúe directamente con el computador. De hecho, para algunos trabajos, tales como el proceso de transacciones, este modo interactivo es fundamental. Al igual que la multiprogramación permite al procesador manejar varias tareas por lotes al mismo tiempo, la multiprogramación puede también utilizarse para manejar varias tareas interactivas, en este último caso, la técnica se conoce como tiempo compartido, porque refleja el hecho de que el tiempo del procesador es compartido entre los diversos usuarios. La técnica básica de un sistema de tiempo compartido es tener a varios usuarios utilizando simultáneamente el sistema mediante terminales, mientras que el sistema operativo intercala la ejecución de cada programa de usuario en ráfagas cortas de computo o cuantos (quantum); de esta manera, si hay n usuarios que solicitan servicio a la vez, cada usuario solo dispondrá, en promedio, de $1/n$ de la atención efectiva del computador, sin contar con la sobrecarga del sistema operativo, sin embargo, dado el tiempo de reacción relativamente lento que tiene el ser humano, el tiempo de respuesta en un sistema correctamente diseñado debería ser comparable al de un computador dedicada.

El tiempo compartido y la multiprogramación plantean una multitud de problemas nuevos para el sistema operativo. Si hay varios trabajos en memoria, entonces deben protegerse de injerencias unos de otros, como, por ejemplo, que uno modifique los datos de otro. Con varios usuarios interactivos, el sistema de archivos debe protegerse de forma que solo los usuarios autorizados puedan tener acceso a un archivo en particular.

Tipos de SO: existen varios tipos de SO, algunos de ellos son:

- **Paralelos:** se pretende que cuando existan dos o más procesos que compitan por algún recurso se puedan realizar o ejecutar al mismo tiempo, es un sistema con más de una CPU, por lo que es conocido como Sistemas Multiprocesador. Los procesadores comparten la memoria y el reloj, dando mayor productividad e incrementando el tiempo de procesamiento.
- **Distribuidos:** permiten distribuir trabajos, tareas o procesos, entre un conjunto de procesadores. Puede ser que este conjunto de procesadores esté en un equipo o en diferentes, en este caso es transparente para el



usuario. Los sistemas distribuidos deben de ser muy confiables, ya que si un componente del sistema se descompone otro componente debe de ser capaz de reemplazarlo. En estos cada procesador cuenta con su propia memoria local, la comunicación se da sobre líneas de comunicación, sus ventajas son la de compartir recursos, aumento de la productividad y confiabilidad.

- **Tiempo real:** son aquellos en los cuales no tiene importancia el usuario, sino los procesos, por lo general, están subutilizados sus recursos con la finalidad de prestar atención a los procesos en el momento que lo requieran. Se utilizan en entornos donde son procesados un gran número de sucesos o eventos. Por lo general se utilizan para controlar dispositivos de aplicaciones delicadas como experimentos científicos, médicos, industria, etc. Hay restricciones de tiempo que se deben cumplir y respetar.
- **Portables:** están destinados para aparatos tecnológicos móviles, celulares, tablets, etc. Contienen PDA, su memoria es limitada, los procesadores son más lentos, con relación a los previamente desarrollados, y sus pantallas son más pequeñas.

Relación del SO con el hardware: los SO modernos están controlados mediante interrupciones, si no hay ningún proceso que ejecutar, ningún dispositivo de E/S al que dar servicio y ningún usuario al que responder, un sistema operativo debe permanecer inactivo, esperando a que algo ocurra, los sucesos casi siempre se indican mediante la ocurrencia de una interrupción o una excepción, una excepción es una interrupción generada por software, debida a un error (por ejemplo, una división por cero o un acceso a memoria no válido) o a una solicitud específica de un programa de usuario de que se realice un servicio del sistema operativo. La característica de un sistema operativo de estar controlado mediante interrupciones define la estructura general de dicho sistema, para cada tipo de interrupción, diferentes segmentos de código del sistema operativo determinan qué acción hay que llevar a cabo., por lo que se utilizará una rutina de servicio a la interrupción (RAI) que es responsable de tratarla. Dado que el sistema operativo y los usuarios comparten los recursos hardware y software del sistema informático, necesitamos asegurar que un error que se produzca en un programa de usuario sólo genere problemas en el programa que se estuviera ejecutando, con la compartición, muchos procesos podrían verse afectados negativamente por un fallo en otro programa.

Para asegurar la correcta ejecución del sistema operativo, tenemos que poder distinguir entre la ejecución del código del sistema operativo y del código definido por el usuario, el método que usan la mayoría de los sistemas informáticos consiste en proporcionar soporte hardware que nos permita diferenciar entre varios modos de ejecución, como mínimo, necesitamos dos modos diferentes de operación: modo usuario y modo kernel o supervisor. Un bit, denominado bit de modo, se añade al hardware de la computadora para indicar el modo actual: *kernel* (0) o usuario (1), con el bit de modo podemos diferenciar entre una tarea que se ejecute en nombre del sistema operativo y otra que se ejecute en nombre del usuario, cuando el sistema informático está ejecutando una aplicación de usuario, el sistema se encuentra en modo de usuario, sin embargo, cuando una aplicación de usuario solicita un servicio del sistema operativo, debe pasar del modo de usuario al modo kernel para satisfacer la solicitud. Cuando se arranca el sistema, el hardware se inicia en el modo kernel, el sistema operativo se carga y se inician las aplicaciones de usuario en el modo usuario, cuando se produce una excepción o interrupción, el hardware comuta del modo de usuario al modo kernel, en consecuencia, cuando el sistema operativo obtiene el control de la computadora, estará en el modo kernel, el sistema siempre cambia al modo de usuario antes de pasar el control a un programa de usuario. El modo dual de operación nos proporciona los medios para proteger el sistema operativo de los usuarios que puedan causar errores, y también para proteger a los usuarios de los errores de otros usuarios. La falta de un modo dual soportado por

hardware puede dar lugar a serios defectos en un SO, por ejemplo el MS-DOS escrito por Intel 8088, no disponía de este modo, por lo tanto un programa de usuario que se ejecute erróneamente puede corromper el SO al mismo tiempo, ya que pueden acceder directamente a las funciones básicas de E/S para escribir directamente a la pantalla o al disco. En Windows 2000 el modo kernel ejecuta los servicios ejecutivos, mientras que el modo usuario ejecuta los procesos de usuario; cuando un programa se bloque en modo usuario, a lo sumo se escribe un suceso en el registro de sucesos, si se bloque estando en modo kernel se produce la BSOD, la pantalla azul de la muerte, y el servidor se detiene. En resumen:

I. **Modo kernel:**

- Modo privilegiado.
- Manejo estricto de pautas de confiabilidad/seguridad.
- Manejo de:
 - CPU, memoria y E/S.
 - Administración multiprocesador, diagnósticos y testing.
 - Partes del file-system y la interface de red.

II. **Modo usuario:**

- Es más flexible.
- Funciones de mantenimiento más simples, debugging:
 - Compiler, assembler, interpreter, linker/loader.
 - File-system management, telecommunication, network management.
 - Editors, spreadsheets, user applications.

Protección de la memoria: la memoria principal y los registros integrados dentro del propio procesador son las únicas áreas de almacenamiento a las que la CPU puede acceder directamente, hay instrucciones de máquina que toman como argumentos direcciones de memoria, pero no existe ninguna instrucción que acepte direcciones de disco, por lo tanto, todas las instrucciones en ejecución y los datos utilizados por esas instrucciones deberán encontrarse almacenados en uno de esos dispositivos de almacenamiento de acceso directo. Como la memoria alberga tanto al SO como a los usuarios debemos garantizar una correcta operación que proteja al SO de los posibles accesos por parte de los procesos de los usuarios y que también proteja a unos procesos de usuarios de otros, esta protección debe ser proporcionada por el hardware y puede implementarse de diversas formas una de ellas es asegurarnos de que cada proceso disponga de un espacio de memoria separado, para hacer esto, debemos poder determinar el rango de direcciones legales a las que el proceso pueda acceder y garantizar también que el proceso sólo acceda a esas direcciones legales, podemos proporcionar esta protección utilizando dos registros, usualmente una base y un límite. El registro base almacena la dirección de memoria física legal más pequeña, mientras que el registro límite especifica el tamaño del rango.

La protección del espacio de memoria se consigue haciendo que el hardware de la CPU compare todas las direcciones generadas en modo usuario con el contenido de esos registros, cualquier intento, por parte de un programa que se esté ejecutando en modo usuario, de acceder a la memoria del sistema operativo o a la memoria de otros usuarios hará que se produzca una interrupción hacia el sistema operativo, que tratará dicho intento como un error fatal. Los registros base y límite sólo pueden ser cargados por el sistema operativo, que utiliza una instrucción privilegiada especial, puesto que las instrucciones privilegiadas sólo pueden ser ejecutadas en modo kernel y como sólo el sistema operativo se ejecuta en modo kernel, únicamente el sistema operativo podrá cargar los registros base y límite, este esquema permite al sistema operativo modificar el valor de los registros, pero evita que los programas de usuario cambien el contenido de esos registros.

Protección de las E/S: se logra al no permitir que los programas actúen directamente sobre los dispositivos, sino a través de llamadas a los manejadores de dispositivos que forman parte del sistema de operación. De esta forma se puede chequear si la solicitud es correcta o no y evitar que algo vaya mal. Para evitar que un programa opere directamente con la E/S, las instrucciones correspondientes se declaran como privilegiadas y por ello sólo podrán ser utilizadas por parte del SO, es decir, que el hardware deberá brindar dualidad en el modo en que los programas se ejecutan. El primero es el modo "kernel" y el segundo es el modo usuario. El SO correrá en modo protegido, con derecho a usar instrucciones privilegiadas y todos los demás en modo usuario.

Protección de la CPU: debemos asegurar que el SO mantenga el control sobre la CPU, para alcanzar este objetivo, podemos usar un temporizador, el cual puede configurarse para interrumpir a la computadora después de un período especificado, este periodo puede ser fijo o variable. Generalmente, se implementa un temporizador variable mediante un reloj de frecuencia fija y un contador, el SO configura el contador, cada vez que el reloj avanza, el contador se decremente, cuando el contador alcanza el valor 0, se produce una interrupción. Antes de devolver el control al usuario, el SO se asegura de que el temporizador esté configurado para realizar interrupciones, cuando el temporizador interrumpe, el control se transfiere automáticamente al SO, que puede tratar la interrupción como un error fatal o puede conceder más tiempo al programa, evidentemente, las instrucciones que modifican el contenido del temporizador son instrucciones privilegiadas. Por lo tanto, podemos usar el temporizador para impedir que un programa de usuario se esté ejecutando durante un tiempo excesivo, una técnica sencilla consiste en inicializar un contador con la cantidad de tiempo que esté permitido que se ejecute un programa. Las instrucciones que modifican el funcionamiento del temporizador son privilegiadas, al contador se le asigna el valor del proceso que se quiere ejecutar.

Interacción de usuario del SO: existen dos métodos fundamentales para que los usuarios interactúen con el sistema operativo, una técnica consiste en proporcionar una interfaz de línea de comandos o intérprete de comandos, que permita a los usuarios introducir directamente comandos que el sistema operativo pueda ejecutar. El segundo método permite que el usuario interactúe con el sistema operativo a través de una interfaz gráfica de usuario o GUI.

Llamadas al sistema (System Calls): es el mecanismo usado por una aplicación para solicitar un servicio al SO, comúnmente usan una instrucción especial de la CPU que causa que procesador transfiera el control a un código privilegiado, esto permite que al código privilegiado especificar donde va a ser conectado así como el estado del procesador. Cuando una llamada al sistema es invocada, la ejecución del programa que invoca es interrumpida y sus datos son guardados para poder continuar ejecutándose luego. El procesador entonces comienza a ejecutar las instrucciones de código de alto nivel de privilegio, para realizar la tarea requerida, cuando esta finaliza, se retorna al proceso original y continúa su ejecución. De esta forma los programas más sencillos pueden hacer un uso intensivo del SO, sin embargo, la mayoría de los programadores no ven este nivel de detalle, ya que normalmente los desarrolladores de aplicaciones diseñan sus programas utilizando una API, la cual especifica un conjunto de funciones que el programador puede usar. Las llamadas al sistema pueden agruparse de forma muy general en cinco categorías principales:

I. **Control de procesos.**

- Terminar, abortar.
- Cargar, ejecutar.

- Crear procesos, terminar procesos.
 - Obtener atributos del proceso, definir atributos del proceso.
 - Esperar para obtener tiempo.
 - Esperar suceso, señalizar suceso.
 - Asignar y liberar memoria.
- II. Manejo de archivos.**
- Crear archivos, borrar archivos.
 - Abrir, cerrar.
 - Leer, escribir, reposicionar.
 - Obtener atributos de archivo, definir atributos de archivo.
- III. Manejo de dispositivos.**
- Solicitar dispositivos, liberar dispositivo.
 - Leer, escribir, reposicionar.
 - Obtener atributos de dispositivo, definir atributos de dispositivo.
 - Conectar y desconectar dispositivos lógicamente.
- IV. Mantenimiento de información del sistema.**
- Obtener la hora o la fecha, definir la hora o la fecha.
 - Obtener datos del sistema, establecer datos del sistema.
 - Obtener atributos de procesos, archivos o dispositivos.
 - Establecer los atributos de procesos, archivos o dispositivos.
- V. Comunicaciones.**
- Crear, eliminar conexiones de comunicación.
 - Enviar, recibir mensajes.
 - Transferir información de estado.
 - Conectar y desconectar dispositivos remotos.

Las llamadas al sistema trabajan de forma diferente en cada tipo de SO, por ejemplo:

- **Linux 2.6:** se emite una interrupción para invocar al Kernel, se llama al Interruption Handler, y el proceso indica con un número la llamada al sistema que desea invocar, cada llamada al sistema tiene un número asignado dentro del kernel, este valor nunca debe ser alterado ni reutilizado en el caso que desaparezca una llamada al sistema.
- **Windows:** las funciones que conforman una API invocan, habitualmente, a las llamadas al sistema por cuenta del programador de la aplicación, por lo tanto la API de Windows está formada por:

I. Funciones de la API.

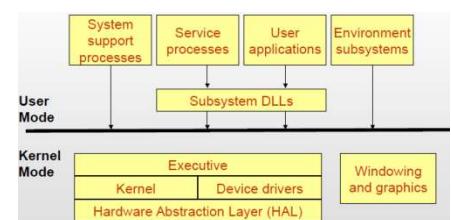
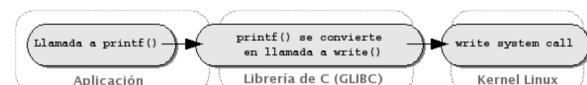
- Subrutinas invocables y documentadas.
- Crear procesos, crear archivos, recibir mensajes.

II. Servicios del sistema.

- Funciones no documentadas, invocables desde el espacio Usuario.

III. Rutinas internas.

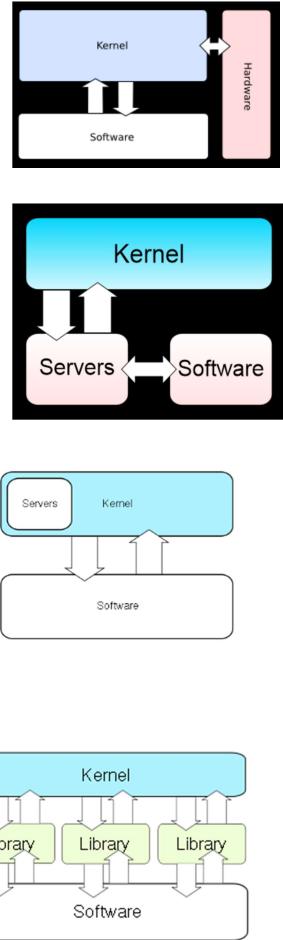
- Subrutinas dentro del kernel.
- Invocables solo del modo kernel.



Tipos de kernel: no necesariamente se necesita un núcleo para usar una computadora. Los programas pueden cargarse y ejecutarse directamente en una computadora vacía, siempre que

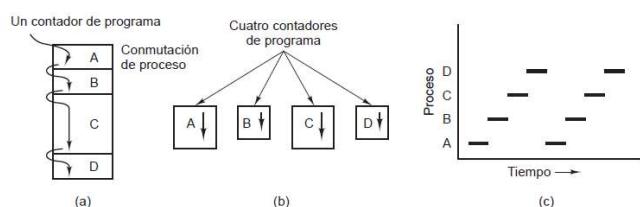
sus autores quieran desarrollarlos sin usar ninguna abstracción del hardware ni ninguna ayuda del sistema operativo. Ésta era la forma normal de usar muchas de las primeras computadoras: para usar distintos programas se tenía que reiniciar y reconfigurar la computadora cada vez, con el tiempo, se empezó a dejar en memoria pequeños programas auxiliares, como el cargador y el depurador, o se cargaban desde memoria de sólo lectura. A medida que se fueron desarrollando, se convirtieron en los fundamentos de lo que llegarían a ser los primeros núcleos de sistema operativo. Hay cuatro grandes tipos de núcleos:

- I. **Monolíticos:** facilitan abstracciones del hardware subyacente realmente potentes y variadas. Frecuentemente se prefieren los núcleos monolíticos frente a los micronúcleos debido al menor nivel de complejidad que comporta el tratar con todo el código de control del sistema en un solo espacio de direccionamiento.
- II. **Micronúcleos (microkernel):** proporcionan un pequeño conjunto de abstracciones simples del hardware, y usan las aplicaciones llamadas servidores para ofrecer mayor funcionalidad. Consiste en definir una abstracción muy simple sobre el hardware, con un conjunto de primitivas o llamadas al sistema que implementan servicios del sistema operativo mínimos, como la gestión de hilos, el espacio de direccionamiento y la comunicación entre procesos. El objetivo principal es la separación de la implementación de los servicios básicos y de la política de funcionamiento del sistema.
- III. **Híbridos:** son muy parecidos a los micronúcleos puros, excepto porque incluyen código adicional en el espacio de núcleo para que se ejecute más rápidamente. Son micronúcleos que tienen algo de código «no esencial» en espacio de núcleo para que éste se ejecute más rápido de lo que lo haría si estuviera en espacio de usuario.
- IV. **Exonúcleos:** no facilitan ninguna abstracción, pero permiten el uso de bibliotecas que proporcionan mayor funcionalidad gracias al acceso directo o casi directo al hardware. La idea subyacente es permitir que el desarrollador tome todas las decisiones relativas al rendimiento del hardware. Los exonúcleos son extremadamente pequeños, ya que limitan expresamente su funcionalidad a la protección y el multiplexado de los recursos.



Procesos: un concepto clave en todos los S.O. es el proceso. Un proceso es en esencia un programa en ejecución. Cada proceso tiene asociado un espacio de direcciones, una lista de ubicaciones de memoria que va desde algún mínimo hasta cierto valor máximo, donde el proceso puede leer y escribir información. El espacio de direcciones contiene el programa ejecutable, los datos del programa y su pila. También hay asociado a cada proceso un conjunto de recursos, que comúnmente incluye registros, PC y stack, entre otros, una lista de archivos abiertos, alarmas pendientes, listas de procesos relacionados y toda la demás información necesaria para ejecutar el programa. En esencia, un proceso es un recipiente que guarda toda la información necesaria para ejecutar un programa.

Modelo del proceso: todo software ejecutable en la computadora, que algunas veces incluye al S.O., se organiza en varios procesos secuenciales. Un proceso no es más que una instancia de un programa en ejecución, incluyendo los valores actuales del PC, los registros y variable. En



concepto, cada proceso tiene su propia CPU virtual; en la realidad, la CPU real conmuta de un proceso a otro, donde esta conmutación rápida de un proceso a otro e conoce como multiprogramación.

La figura "a" muestra una computadora multi-programando cuatro programas en memoria, la "b" lista cuatro procesos, cada uno con su propio flujo de control, es decir, su propio contador de programa lógico, y cada uno ejecutándose en forma independiente. Desde luego hay un solo PC físico, por lo que cuando se ejecuta cada proceso, se carga su contador de programa lógico en el PC real. Cuando termina, por el tiempo que tenga asignado, el contador físico se guarda en el lógico almacenado, del proceso en memoria. En la figura "c" podemos ver que durante un intervalo suficientemente largo todos los procesos han progresado, pero en cualquier momento dado sólo hay un proceso en ejecución.

Componentes de un proceso: al ser una entidad de abstracción, tiene que contener unos componentes para poder ejecutarse. Como mínimo un proceso debe contar con:

- **Sección de código:** el código se encuentra de forma textual. Alguien lo carga en memoria y se crea el proceso.
- **Sección de datos:** contiene las variables iniciales o globales.
- **Stacks:** son datos temporarios, parámetros, variables temporales y direcciones de retorno. Un proceso cuenta con 1 o más stacks, en general dos, Usuario y Kernel. Estos se crean automáticamente y su medida se ajusta en run time. Está formado por stack frames que son pushed, al llamar a una rutina, y Popped, cuando se retorna de ella. El stack frame tiene los parámetros de la rutina, variables locales, y datos necesarios para recuperar el stack frame anterior, el PC y el valor de stack pointer en el momento llamado.

Un proceso a su vez contiene atributos, una ID, identificación de proceso, identificación del proceso padre. Todos los procesos se crean a través de un proceso, salvo el primero que no nace de nadie. También cuenta con información del usuario que lo "disparó", si hay una estructura de grupos, contiene información del grupo que lo disparó. Esto permite darle seguridad a un proceso ya que sabrá quién ejecutó/disparó el proceso. En el sistema multi-usuario te permite saber las terminales que están en ejecución y cuáles tienen acceso para trabajar sobre el proceso.

Para implementar el modelo de procesos el S.O. mantiene una tabla llamada tabla de procesos, con sólo una entrada por cada proceso, PCB. Esta entrada contiene información importante acerca del estado del proceso, incluyendo su PC, apuntador de pila, asignación de memoria, estado de sus archivos abiertos, información de contabilidad y planificación, y todo lo demás que debe guardarse acerca del proceso cuando éste cambia del estado en ejecución a listo o bloqueado, de manera que se pueda reiniciar posteriormente como si nunca se hubiera detenido.

La figura muestra algunos de los campos clave en un sistema típico. Los campos en la primera columna se relacionan con la administración de procesos; los otros dos se relacionan con la administración de memoria y archivos, respectivamente. Hay que recalcar que los campos contenidos en la tabla de procesos

Administración de procesos Registros Contador del programa Palabra de estado del programa Apuntador de la pila Estado del proceso Prioridad Parámetros de planificación ID del proceso Proceso padre Grupo de procesos Señales Tiempo de inicio del proceso Tiempo utilizado de la CPU Tiempo de la CPU utilizado por el hijo Hora de la siguiente alarma	Administración de memoria Apuntador a la información del segmento de texto Apuntador a la información del segmento de datos Apuntador a la información del segmento de pila	Administración de archivos Directorio raíz Directorio de trabajo Descripciones de archivos ID de usuario ID de grupo
--	--	---

varían de un sistema a otro, pero esta figura nos da una idea general de los tipos de información necesaria.

Contexto de un proceso: es una definición teórica, la cual incluye a toda la información que necesita la CPU para poder utilizar el proceso. Son parte del contexto, los registros de CPU, inclusive el PC, prioridad del proceso, si tiene E/S pendientes, etc.

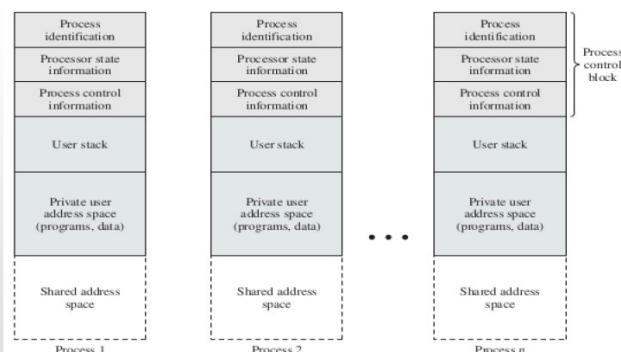
Las interrupciones hacen que el S.O. obligue a la CPU a abandonar su tarea actual, para ejecutar una rutina del kernel. Estos sucesos se producen con frecuencia en los sistemas de propósito general. Cuando se produce un interrupción el sistema tiene que guardar el contexto actual del proceso que se está ejecutando en la CPU, de modo que puede restaurar dicho contexto cuando su prosesamiento concluya, suspendiendo el proceso y reanudándolo después. El contexto se almacena en el PCB del proceso e incluye el valor de los registros de la CPU, el estado del proceso y la información de gestión de memoria.

El tiempo dedicado al cambio de contexto e tiempo desperdiciado, dado que el sistema no realiza ningún trabajo útil durante la conmutación. La velocidad del cambio de contexto varía de una máquina a otra, dependiendo de la velocidad de memoria, del N° de registros que tengan que copiarse y de la existencia de instrucciones especiales, por lo general son unos milisegundos. De todos modos este tiempo dependerá del soporte que nos de el hardware.

Espacio de direcciones de un proceso: es el conjunto de direcciones de memoria que ocupa un proceso. Las direcciones de memoria del tipo lógico son creadas y utilizadas por los procesos en el sistema, estas a cambio de las físicas pueden sufrir una serie de cambios o transformaciones por la CPU antes de que sean convertidas. Cada proceso cargado en memoria tiene su espacio de memoria asignado en el sistema, los cuales se dividen en dos tipos:

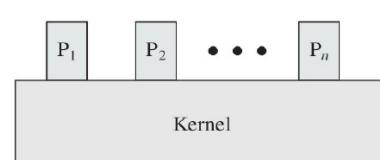
- I. **Modo usuario:** puede acceder sólo a su espacio de direcciones, que son específicas de cada usuario.
 - II. **Modo Kernel:** en el contexto de un proceso, se puede acceder a estructuras internas o a espacios de direcciones de otros procesos.

The diagram illustrates the memory organization in a multi-process system. It shows two vertical columns, each representing a process. The top section of each column is labeled 'Private user address space (programs, data)'. Below this, a dashed box labeled 'Shared address space' spans both columns. At the bottom of each column is the label 'Process 1' and 'Process 2' respectively.



En todo este espacio de direcciones se debe encontrar el S.O., el cual es un conjunto de módulos de software y se ejecuta en el procesador como cualquier otro proceso. Pero no es un proceso, ya que no se comporta como tal. Se han desarrollado dos enfoques de como introducir el S.O. dentro de este espacio de direcciones:

- I. **Kernel como entidad independiente:** se ejecuta fuera de todo proceso. El crea a todos los demás, teniendo su propia región de memoria, su propio stack. Cuando un



proceso es interrumpido o realiza una System Call, el contexto del proceso se salva y el control se pasa al Kernel. Teniendo así un solo stack en modo Kernel. Una vez finalizada su actividad, le devuelve el control al proceso o a otro diferente.

Como problemas presenta si un proceso llama a una System Call tiene que resolver toda la interrupción de modo Kernel y luego volver a otro proceso porque existe un solo stack en modo Kernel. Si otro proceso necesita hacer algo en modo Kernel, no va a poder porque ya está siendo utilizado.

- II. **Kernel dentro del proceso:** el código del Kernel se encuentra dentro del espacio de direcciones de cada proceso, está dentro del contexto de estos. Se tiene un solo código físico pero virtualmente está en los procesos y se ejecuta en el mismo contexto que algún proceso de usuario.

El Kernel se puede ver como una colección de rutinas que el proceso utiliza, con lo cual no hay un solo stack Kernel; si no que todos los procesos contienen dos stack uno de modo usuario y otro de modo Kernel, cambiando solamente el modo.

Cuando ocurre una interrupción se pasa de código al Kernel y luego se vuelve al mismo proceso.

Ahora se puede referenciar la memoria como de modo usuario o modo privilegiado, para que no se pueda ejecutar todo en uno.

La desventaja que trae este enfoque, es que se ocupa más memoria. Pero en la actualidad eso no es un problema.

Estados de un proceso: a medida que se ejecuta un proceso, el proceso va cambiando de estado.

El estado de un proceso se define, en parte, según la actividad actual de dicho proceso. Cada proceso puede estar en uno de los siguientes estados:

- **Nuevo, new:** el proceso está siendo creado.
- **En ejecución, running:** se están ejecutando instrucciones.
- **En espera, waiting:** el proceso está esperando a que se produzca un suceso, como la terminación de una operación de E/S o la recepción de una señal.
- **Preparado, ready:** el proceso está a la espera de que le asignen a un procesador.
- **Terminado, terminated:** ha terminado la ejecución del proceso.



Los estados que representan se encuentran en todos los sistemas. Determinados S.O. definen los estados de los procesos de forma más específica. Es importante reconocer que sólo puede haber un proceso ejecutándose en cualquier procesador en cada instante concreto. Sin embargo, puede haber muchos procesos preparados y en espera.

Colas de planificación: a medida que los procesos entran en el sistema, se colocan en una cola de trabajos que contienen todos los procesos del sistema. Los procesos que residen en la memoria principal y están preparados y en espera de ejecutarse se mantienen en una lista denominada cola de procesos preparados cola de procesos preparados. Generalmente, esta cola se almacena en forma de lista enlazada. La cabecera de la cola de procesos preparados contiene punteros al

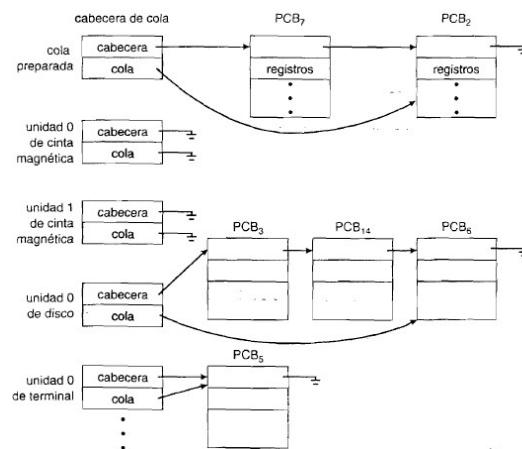
primer y último bloques de control de procesos, PCB, de la lista. Cada PCB incluye un campo de puntero que apunta al siguiente PCB de la cola de procesos preparado.

El sistema también incluye otras colas. Cuando se asigna la CPU a un proceso, éste se ejecuta durante un rato y finalmente termina, es interrumpido o espera a que se produzca un determinado suceso, como la terminación de una solicitud de E/S. Dado que hay muchos procesos en el sistema, el disco puede estar ocupado con la solicitud de E/S de algún otro proceso. Por tanto, nuestro proceso puede tener que esperar para poder acceder al disco. La lista de procesos en espera de un determinado dispositivo de E/S se denomina cola del dispositivo. Cada dispositivo tiene su propia cola.

Hay dos tipos de colas: la de procesos preparados y un conjunto de colas de dispositivo. Cada proceso nuevo se coloca inicialmente en la cola de procesos preparados, donde espera hasta que es seleccionado para ejecución, es decir, hasta que es despachado. Una vez que se asigna la CPU al proceso y éste comienza a ejecutarse, se puede producir uno de los siguientes sucesos:

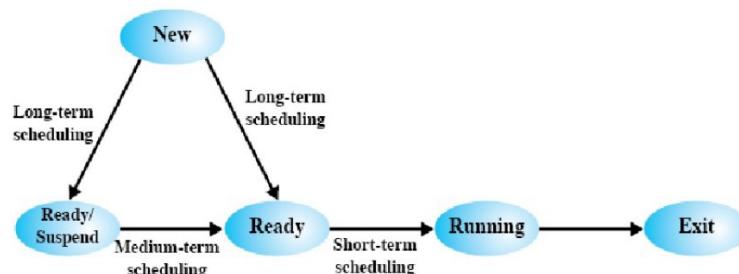
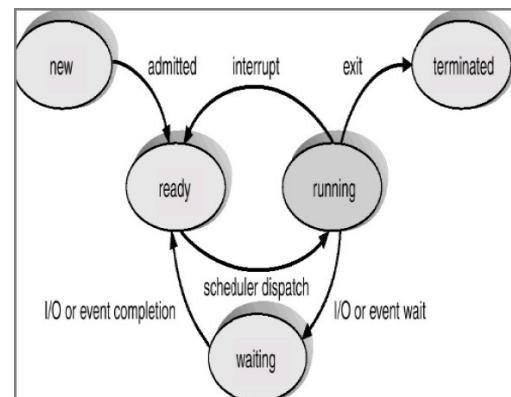
- El proceso podría ejecutar una solicitud de E/S y ser colocado, como consecuencia, en una cola de E/S.
- El proceso podría crear un nuevo subproceso y esperar a que éste termine.
- El proceso podría ser desalojado de la CPU como resultado de una interrupción y puesto de nuevo en la cola de procesos preparados.

En los dos primeros casos, el proceso terminará, por cambiar del estado de espera al de preparado y será colocado de nuevo en la cola de procesos preparados. Los procesos siguen este ciclo hasta que termina su ejecución, momento en el que se elimina el proceso de todas las colas y se designan su PCB y sus recursos.



Módulos de la planificación: durante su tiempo de vida, los procesos se mueven entre las diversas colas de planificación. El S.O., como parte de la tarea de planificación, debe seleccionar de alguna manera los procesos que se encuentran en estas colas. El proceso de selección se realiza mediante un planificador apropiado.

A menudo, en un sistema de procesamiento por lotes, se envían más procesos de los que se pueden ser ejecutados de forma inmediata. Estos procesos se guardan en cola en un dispositivo de almacenamiento masivo, donde se mantienen para su posterior ejecución. El planificador a largo plazo o long term scheduler, selecciona procesos de esta cola y los carga en memoria para su ejecución. El planificador a



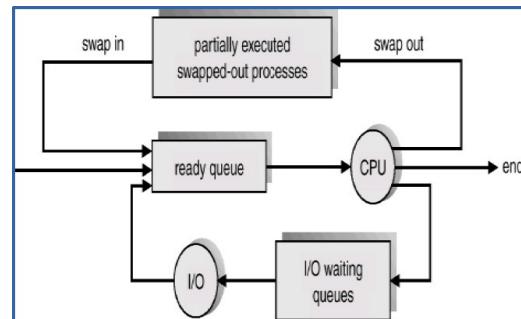
corto plazo o short term scheduler, selecciona de entre los procesos que ya están preparados para ser ejecutados y asigna la CPU a uno de ellos.

La principal diferencia entre estos dos planificadores se encuentra en la frecuencia de ejecución. El planificador a corto plazo debe seleccionar un nuevo proceso para la CPU frecuentemente. Un proceso puede ejecutarse sólo durante unos pocos milisegundos antes de tener que esperar por una solicitud de E/S.

El planificador a largo plazo se ejecuta mucho menos frecuentemente; pueden pasar minutos entre la creación de un nuevo proceso y el siguiente. El planificador a largo plazo controla el grado de multiprogramación. Si el grado es estable, entonces la tasa promedio de creación de procesos debe ser igual a la tasa promedio de salida de procesos del sistema. Por tanto, el planificador a largo plazo puede tener que invocarse sólo cuando un proceso abandona el sistema. Puesto que el intervalo entre ejecuciones es más largo, el planificador a largo plazo puede permitirse emplear más tiempo en decidir qué proceso debe seleccionarse para ser ejecutado.

Es importante que el planificador a largo plazo haga una elección cuidadosa. En general, la mayoría de los procesos pueden describirse como limitados por la E/S o limitados por la CPU. Un proceso limitado por E/S es aquel que invierte la mayor parte de su tiempo en operaciones de E/S en lugar de en realizar cálculos. Por el contrario, un proceso limitado por la CPU genera solicitudes de E/S con poco frecuencia, usando la mayor parte de su tiempo en realizar cálculos. Es importante que el planificador a largo plazo seleccione una adecuada mezcla de procesos, equilibrando los procesos limitados por E/S y los procesos limitados por la CPU. Si todos los procesos son limitados por la E/S, la cola de procesos preparados casi siempre estará vacía y el planificador a corto plazo tendrá poco que hacer. Si todos los procesos son limitados por la CPU, la cola de espera de E/S casi siempre estará vacía, los dispositivos apenas se usarán, y de nuevo el sistema se desequilibrará. Para obtener un mejor rendimiento, el sistema dispondrá entonces de una combinación equilibrada de procesos limitados por la CPU y de procesos limitados por E/S. En algunos sistemas, el planificador a largo plazo puede no existir o ser mínimo. Por ejemplo, los sistemas de tiempo compartido, tales como UNIX y los sistemas Microsoft Windows, a menudo no disponen de planificador a largo plazo, sino que simplemente ponen todos los procesos nuevos en memoria para que los gestione el planificador a corto plazo. La estabilidad de estos sistemas depende bien de una limitación física, tal como el número de terminales disponibles. Si el rendimiento desciende a niveles inaceptables en un sistema multiusuario, algunos usuarios simplemente lo abandonarán.

Algunos sistemas operativos, como los sistemas de tiempo compartido, pueden introducir un nivel intermedio adicional de planificación; en la figura se muestra este planificador. La idea clave subyacente a un planificador a medio plazo es que, en ocasiones, puede ser ventajoso eliminar procesos de la memoria, con lo que dejan de contender por la CPU, y reducir así el grado de multiprogramación. Después, el proceso puede volver a cargarse en memoria, continuando su ejecución en el punto en que se interrumpió. Este esquema se denomina intercambio. El planificador a medio plazo descarga y luego vuelve a cargar el proceso, es decir swapea el proceso o lo pone en espera. El intercambio puede ser necesario para mejorar la mezcla de procesos o porque un cambio en los requisitos de memoria haya hecho que se sobrepase la memoria disponible, requiriendo que se libere memoria.



En resumen, los estados por los que pasa un proceso, nuevo, listo, suspendido, ejecución y terminado:

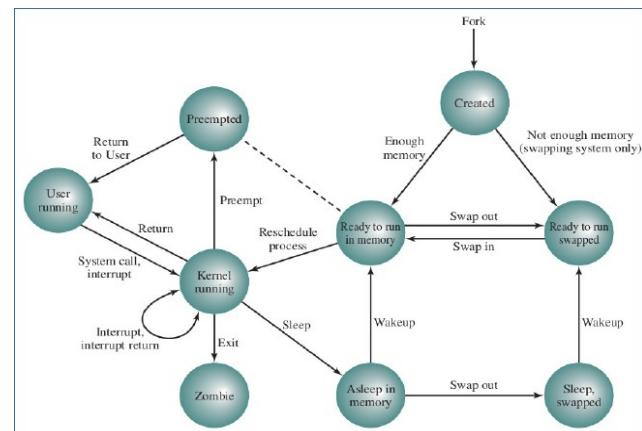
- **Sobre el estado nuevo:** un usuario “dispara” el proceso. Un proceso es creado por otro proceso: su proceso padre. En este estado se crean las estructuras asociadas, y el proceso queda en la cola de procesos, normalmente en espera de ser cargado en memoria.
- **Sobre el estado listo:** luego de que el scheduler de largo plazo eligió al proceso para cargarlo en memoria, el proceso queda en estado de listo. Esté sólo necesita que se le asigne CPU, quedando almacenado en la cola de procesos listos, ready queue.
- **Sobre el estado en ejecución:** el scheduler de corto plazo lo eligió para asignarle CPU. Esté tendrá la CPU hasta que se termine el período de tiempo asignado, quantum o time slice, termine o hasta que necesite realizar alguna operación de E/S.
- **Sobre el estado de espera:** el proceso necesita que se cumpla el evento esperando para continuar. El evento puede ser la terminación de una E/S solicitada, o la llegada de una señal por parte de otro proceso, sigue en memoria pero no tiene la CPU. Al cumplirse el evento, pasará al estado listo.

Las transiciones por las que pasan los procesos son:

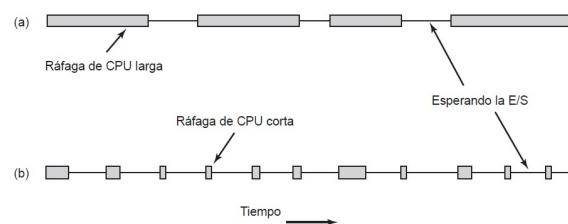
- **New-ready:** por elección del scheduler de largo plazo, carga en memoria.
- **Ready-running:** por elección del scheduler de corto plazo, asignación de CPU.
- **Running-waiting:** el proceso “se pone a dormir” esperando por un evento.
- **Waiting-ready:** terminó la espera y compite nuevamente por la CPU.
- **Running-ready:** este es un caso especial, cuando el proceso termina su quantum, franja de tiempo, sin haber necesitado ser interrumpido por un evento, pasa al estado de ready, para competir por CPU, pues no está esperando por ningún evento.

Diagrama incluyendo swapping: explicación por cada uno de los estados:

- I. Ejecución en modo user.
- II. Ejecución en modo kernel.
- III. El proceso está listo para ser ejecutado cuando sea elegido.
- IV. Proceso en espera en memoria principal.
- V. Proceso listo, pero el swapper debe llevar al proceso a memoria pero antes que el kernel lo pueda elegir para ejecutar.
- VI. Proceso en espera en memoria secundaria.
- VII. Proceso retornando desde el modo kernel al user. Pero el kernel se apropiá, hace un context switch para darla la CPU a otro proceso.
- VIII. Proceso recientemente creado y en transición: existe, pero aún no está listo para ejecutar, ni está dormido.
- IX. El proceso ejecutó la system call exit y está en estado zombie. Ya no existe más, pero se registran datos sobre su uso, código resultante del exit. Es el estado final.



Comportamiento de un proceso: Casi todos los procesos alternan ráfagas de cálculos con peticiones de E/S, de disco. Por lo general la CPU opera durante cierto tiempo sin detenerse, después se realiza una llamada al sistema para leer datos de un archivo o escribirlos en el mismo. Cuando se



completa la llamada al sistema, la CPU realiza cálculos de nuevo hasta que necesita más datos o tiene que escribir más datos y así sucesivamente. Hay que tener en cuenta que algunas actividades de E/S cuentan como cálculos.

Algunos procesos, como el que se muestra en "a", invierten la mayor parte de su tiempo realizando cálculos, mientras que otros, como el que se muestra en "b", invierten la mayor parte de su tiempo esperando la E/S. A los primeros se les conoce como limitados a cálculos; a los segundos como limitados a E/S, I/O-bound. Por lo general, los procesos limitados a cálculos tienen ráfagas de CPU largas y en consecuencia, esperas infrecuentes por la E/S, mientras que los procesos limitados a E/S tienen ráfagas de CPU cortas y por ende, esperas frecuentes por la E/S. Hay que observar que el factor clave es la longitud de la ráfaga de CPU, no de la ráfaga de E/S. Los procesos limitados a E/S están limitados a la E/S debido a que no realizan muchos cálculos entre una petición de E/S y otra, no debido a que tengan peticiones de E/S en especial largas. Se requiere el mismo tiempo para emitir la petición de hardware para leer un bloque de disco, sin importar qué tanto, o qué tan poco, tiempo se requiera para procesar los datos, una vez que lleguen. Vale la pena observar que, a medida que las CPUs se vuelven más rápidas, los procesos tienden a ser más limitados a E/S. Este efecto ocurre debido a que las CPUs están mejorando con mucha mayor rapidez que los discos.

Planificación: cuando una computadora se multiprograma, con frecuencia tiene varios procesos o hilos que compiten por la CPU al mismo tiempo. Esta situación ocurre cada vez que dos o más de estos procesos se encuentran al mismo tiempo en el estado listo. Si sólo hay una CPU disponible, hay que decidir cuál proceso se va a ejecutar a continuación. La parte del sistema operativo que realiza esa decisión se conoce como planificador de procesos y el algoritmo que utiliza se conoce como algoritmo de planificación. Estos términos conforman el tema a tratar en las siguientes secciones. Muchas de las mismas cuestiones que se aplican a la planificación de procesos también se aplican a la planificación de hilos, aunque algunas son distintas. Cuando el kernel administra hilos, por lo general la planificación se lleva a cabo por hilo, e importa muy poco, o nada, a cuál proceso pertenece ese hilo.

Algoritmos de planificación: los algoritmos de planificación se pueden dividir en dos categorías con respecto a la forma en que manejan las interrupciones del reloj:

- **No apropiativo, nonpreemptive:** selecciona un proceso para ejecutarlo y después sólo deja que se ejecute hasta que el mismo se bloquee, ya sea en espera de una operación de E/S o de algún otro proceso, o hasta que libera la CPU en forma voluntaria. Incluso aunque se ejecute durante horas, no se suspenderá de manera forzosa. En efecto, no se toman decisiones de planificación durante las interrupciones de reloj. Una vez que se haya completado el procesamiento de la interrupción de reloj, se reanuda la ejecución del proceso que estaba antes de la interrupción, a menos que un proceso de mayor prioridad esté esperando por un tiempo libre que se acabe de cumplir.
- **Apropiativo, preemptive:** selecciona un proceso y deja que se ejecute por un máximo de tiempo fijo. Si sigue en ejecución al final del intervalo de tiempo, se suspende y el planificador selecciona otro proceso para ejecutarlo, si hay uno disponible. Para llevar a cabo la planificación apropiativa, es necesario que ocurra una interrupción de reloj al final del intervalo de tiempo para que la CPU regrese el control al planificador. Si no hay un reloj disponible, la planificación no apropiativa es la única opción.

Categoría de los algoritmos de planificación: no es sorprendente que distintos entornos requieran algoritmos de planificación diferentes. Esta situación se presenta debido a que las diferentes áreas de aplicación, y los distintos tipos de sistemas operativos, tienen diferentes objetivos. En otras

palabras, lo que el planificador debe optimizar no es lo mismo en todos los sistemas. Tres de los entornos que vale la pena mencionar son:

- **Procesamiento por lotes, Batch:** los sistemas de procesamiento por lotes siguen utilizándose ampliamente en el mundo de los negocios. En los sistemas de procesamiento por lotes no hay usuarios que esperen impacientemente en sus terminales para obtener una respuesta rápida a una petición corta. En consecuencia, son aceptables los algoritmos no apropiativos, o apropiativos con largos períodos para cada proceso. Este método reduce la conmutación de procesos y por ende, mejora el rendimiento. En realidad, los algoritmos de procesamiento por lotes son bastante generales y a menudo se pueden aplicar a otras situaciones también, lo cual hace que valga la pena estudiarlos, incluso para las personas que no están involucradas en la computación con mainframes corporativas.
Las metas propias de este tipo de algoritmos son:
 - ✓ **Rendimiento:** maximizar el N° de trabajos por hora.
 - ✓ **Tiempo de retorno:** minimizar los tiempos entre el comienzo y la finalización.
 - ✓ **Uso de la CPU:** mantener la CPU ocupada la mayor cantidad de tiempo posible.Algunos algoritmos de ejemplo son los FCFS, First Come First Seved, o los SJF, Shortest Job First. El primero es una cola y el segundo es el más corto lo trabajo primero.
- **Interactivo:** la apropiación es esencial para evitar que un proceso acapare la CPU y niegue el servicio a los demás. Aun si no hubiera un proceso que se ejecutara indefinidamente de manera intencional, podría haber un proceso que deshabilitara a los demás de manera indefinida, debido a un error en el programa. La apropiación es necesaria para evitar este comportamiento. Los servidores también entran en esta categoría, ya que por lo general dan servicio a varios usuarios, remotos, todos los cuales siempre tienen mucha prisa. Las metas propias de este tipo de algoritmos son:
 - ✓ **Tiempo de respuesta:** responder a peticiones con rapidez.
 - ✓ **Proporcionalidad:** cumplir con expectativas de los usuarios. Si el usuario le pone stop al reproductor de música que la música deje de ser reproducida en un tiempo considerablemente corto.Algunos algoritmos que implementan interactividad son, Round Robin, colas de prioridades, colas multinivel y SRTF, Shortest Remaining Time First.
- **De tiempo real:** la apropiación a veces es no necesaria debido a que los procesos saben que no se pueden ejecutar durante períodos extensos, que por lo general realizan su trabajo y se bloquean con rapidez. La diferencia con los sistemas interactivos es que los sistemas de tiempo real sólo ejecutan programas destinados para ampliar la aplicación en cuestión. Los sistemas interactivos son de propósito general y pueden ejecutar programas arbitrarios que no sean cooperativos, o incluso malintencionados. Las metas a cumplir con estos algoritmos son:
 - ✓ **Cumplir con los plazos cortos:** evitando perder datos.
 - ✓ **Predictibilidad:** evitar la degradación de la calidad en los sistemas bajo distinta circunstancias.

Política contra mecanismo: hemos supuesto tácitamente que todos los procesos en el sistema pertenecen a distintos usuarios y por lo tanto compiten por la CPU. Aunque esto es a menudo cierto, algunas veces sucede que un proceso tiene muchos hijos ejecutándose bajo su control. Por ejemplo, un proceso de un sistema de administración de bases de datos puede tener muchos hijos. Cada hijo podría estar trabajando en una petición distinta o cada uno podría tener cierta función específica que realizar. Es por completo posible que el proceso principal tenga una idea excelente acerca de cuál de sus hijos es el más importante, o que requiere tiempo con más urgencia, y cuál es el menos importante. Por desgracia, ninguno de los planificadores antes

descritos acepta entrada de los procesos de usuario acerca de las decisiones de planificación.

Como resultado, raras veces el planificador toma la mejor decisión.

La solución a este problema es separar el mecanismo de planificación de la política de planificación, un principio establecido desde hace tiempo por Levin y colaboradores, 1975. Esto significa que el algoritmo de planificación está parametrizado de cierta forma, pero los procesos de usuario pueden llenar los parámetros. Considerando el ejemplo de la base de datos.

Suponemos que el kernel utiliza un algoritmo de planificación por prioridad, pero proporciona una llamada al sistema mediante la cual un proceso puede establecer (y modificar) las prioridades de sus hijos. De esta forma, el padre puede controlar con detalle la forma en que se planifican sus hijos, aun y cuando éste no se encarga de la planificación. Aquí el mecanismo está en el kernel, pero la política se establece mediante un proceso de usuario.

Creación de proceso: los sistemas operativos necesitan cierta manera de crear procesos. En sistemas muy simples o sistemas diseñados para ejecutar sólo una aplicación, es posible tener presentes todos los procesos que se vayan a requerir cuando el sistema inicie. No obstante, en los sistemas de propósito general se necesita cierta forma de crear y terminar procesos según sea necesario durante la operación. Ahora analizaremos varias de estas cuestiones. Hay cuatro eventos principales que provocan la creación de procesos:

- I. El arranque del sistema.
- II. La ejecución, desde un proceso, de una llamada al sistema para creación de procesos.
- III. Una petición de usuario para crear un proceso.
- IV. El inicio de un trabajo por lotes.

Generalmente, cuando se arranca un sistema operativo se crean varios procesos. Algunos de ellos son procesos en primer plano; es decir, procesos que interactúan con los usuarios y realizan trabajo para ellos. Otros son procesos en segundo plano, que no están asociados con usuarios específicos sino con una función específica. Por ejemplo, se puede diseñar un proceso en segundo plano para aceptar el correo electrónico entrante, que permanece inactivo la mayor parte del día pero que se activa cuando llega un mensaje. Los procesos que permanecen en segundo plano para manejar ciertas actividades como correo electrónico, páginas Web, noticias, impresiones, etcétera, se conocen como demonios, daemons. Los sistemas grandes tienen comúnmente docenas de ellos.

Además de los procesos que se crean al arranque, posteriormente se pueden crear otros. A menudo, un proceso en ejecución emitirá llamadas al sistema para crear uno o más procesos nuevos, para que le ayuden a realizar su trabajo. En especial, es útil crear procesos cuando el trabajo a realizar se puede formular fácilmente en términos de varios procesos interactivos relacionados entre sí, pero independientes en los demás aspectos. En un multiprocesador, al permitir que cada proceso se ejecute en una CPU distinta también se puede hacer que el trabajo se realice con mayor rapidez.

En la creación del proceso se le debe asignar diferentes atributos como la PCB, asignarle un PID único, asignarle memoria para las regiones de stack, texto y datos, y crear estructuras asociadas. En los sistemas interactivos, los usuarios pueden iniciar un programa escribiendo un comando o haciendo clic en un ícono. Cualquiera de las dos acciones inicia un proceso y ejecuta el programa seleccionado. En los sistemas UNIX basados en comandos que ejecutan X, el nuevo proceso se hace cargo de la ventana en la que se inició. En Microsoft Windows, cuando se inicia un proceso no tiene una ventana, pero puede crear una y la mayoría lo hace. En ambos sistemas, los usuarios pueden tener varias ventanas abiertas a la vez, cada una ejecutando algún proceso. Mediante el ratón, el usuario puede seleccionar una ventana e interactuar con el proceso.

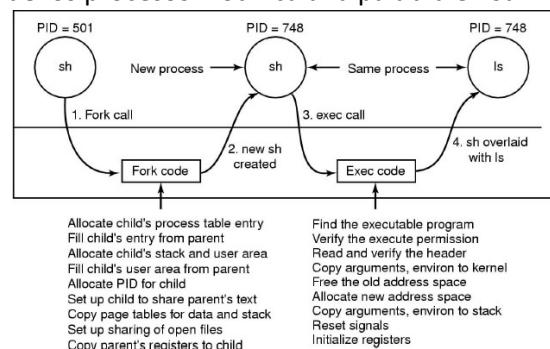
La última situación en la que se crean los procesos se aplica sólo a los sistemas de procesamiento por lotes que se encuentran en las mainframes grandes. Aquí los usuarios pueden enviar trabajos

de procesamiento por lotes al sistema. Cuando el sistema operativo decide que tiene los recursos para ejecutar otro trabajo, crea un proceso y ejecuta el siguiente trabajo de la cola de entrada. Técnicamente, en todos estos casos, para crear un proceso hay que hacer que un proceso existente ejecute una llamada al sistema de creación de proceso. Ese proceso puede ser un proceso de usuario en ejecución, un proceso del sistema invocado mediante el teclado o ratón, o un proceso del administrador de procesamiento por lotes. Lo que hace en todo caso es ejecutar una llamada al sistema para crear el proceso. Esta llamada al sistema indica al sistema operativo que cree un proceso y le indica, directa o indirectamente, cuál programa debe ejecutarlo.

En UNIX sólo hay una llamada al sistema para crear un proceso: fork. Esta llamada crea un clon exacto del proceso que hizo la llamada. Después de fork, los dos procesos, padre e hijo, tienen la misma imagen de memoria, las mismas cadenas de entorno y los mismos archivos abiertos. Por lo general, el proceso hijo ejecuta después a execve o una llamada al sistema similar para cambiar su imagen de memoria y ejecutar un nuevo programa. Por ejemplo, cuando un usuario escribe un comando tal como sort en el shell, éste crea un proceso hijo, que a su vez ejecuta a sort. La razón de este proceso de dos pasos es para permitir al hijo manipular sus descriptores de archivo después de fork, pero antes de execve, para poder lograr la redirección de la entrada estándar, la salida estándar y el error estándar.

Por el contrario, en Windows una sola llamada a una función de Win32, CreateProcess, maneja la creación de procesos y carga el programa correcto en el nuevo proceso. Esta llamada tiene 10 parámetros, que incluyen el programa a ejecutar, los parámetros de la línea de comandos para introducir datos a ese programa, varios atributos de seguridad, bits que controlan si los archivos abiertos se heredan, información de prioridad, una especificación de la ventana que se va a crear para el proceso y un apuntador a una estructura en donde se devuelve al proceso que hizo la llamada la información acerca del proceso recién creado. Además de CreateProcess, Win32 tiene cerca de 100 funciones más para administrar y sincronizar procesos y temas relacionados.

Tanto en UNIX como en Windows, una vez que se crea un proceso, el padre y el hijo tienen sus propios espacios de direcciones distintos. Si cualquiera de los procesos modifica una palabra en su espacio de direcciones, esta modificación no es visible para el otro proceso. En UNIX, el espacio de direcciones inicial del hijo es una copia del padre, pero en definitiva hay dos espacios de direcciones distintos involucrados; no se comparte memoria en la que se pueda escribir, algunas implementaciones de UNIX comparten el texto del programa entre los dos, debido a que no se puede modificar. Sin embargo, es posible para un proceso recién creado compartir algunos de los otros recursos de su creador, como los archivos abiertos. En Windows, los espacios de direcciones del hijo y del padre son distintos desde el principio.



Terminación de procesos: Una vez que se crea un proceso, empieza a ejecutarse y realiza el trabajo al que está destinado. Sin embargo, nada dura para siempre, ni siquiera los procesos. Tarde o temprano el nuevo proceso terminará, por lo general debido a una de las siguientes condiciones:

- I. Salida normal, voluntaria.
- II. Salida por error, voluntaria.
- III. Error fatal, involuntario.
- IV. Eliminado por otro proceso, involuntaria.

La mayoría de los procesos terminan debido a que han concluido su trabajo. Cuando un compilador ha compilado el programa que recibe, ejecuta una llamada al sistema para indicar al sistema operativo que ha terminado. Esta llamada es exit en UNIX y ExitProcess en Windows. Los programas orientados a pantalla también admiten la terminación voluntaria. Los procesadores de palabras, navegadores de Internet y programas similares siempre tienen un ícono o elemento de menú en el que el usuario puede hacer clic para indicar al proceso que elimine todos los archivos temporales que tenga abiertos y después termine. La segunda razón de terminación es que el proceso descubra un error. Por ejemplo, si un usuario escribe el comando "cc foo.c". Para compilar el programa foo.c y no existe dicho archivo, el compilador simplemente termina. Los procesos interactivos orientados a pantalla por lo general no terminan cuando reciben parámetros incorrectos. En vez de ello, aparece un cuadro de diálogo y se le pide al usuario que intente de nuevo.

La tercera razón de terminación es un error fatal producido por el proceso, a menudo debido a un error en el programa. Algunos ejemplos incluyen el ejecutar una instrucción ilegal, hacer referencia a una parte de memoria no existente o la división entre cero. En algunos sistemas, como UNIX, un proceso puede indicar al sistema operativo que desea manejar ciertos errores por sí mismo, en cuyo caso el proceso recibe una señal se interrumpe en vez de terminar.

La cuarta razón por la que un proceso podría terminar es que ejecute una llamada al sistema que indique al sistema operativo que elimine otros procesos. En UNIX esta llamada es kill. La función correspondiente en Win32 es TerminateProcess. En ambos casos, el proceso eliminador debe tener la autorización necesaria para realizar la eliminación. En algunos sistemas, cuando un proceso termina, ya sea en forma voluntaria o forzosa, todos los procesos que creó se eliminan de inmediato también. Sin embargo, ni Windows ni UNIX trabajan de esta forma.

Proceso cooperativos e independientes:

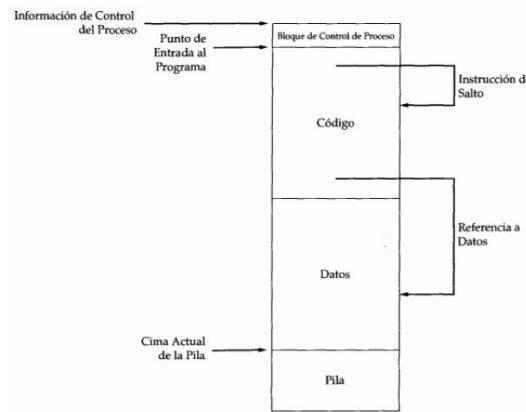
- **Independiente:** el proceso no afecta ni puede ser afectado por la ejecución de otros procesos. No comparte ningún tipo de dato.
- **Cooperativo:** afecta o es afectado por la ejecución de otros procesos en el sistema. Para compartir información, por ejemplo un archivo. Para acelerar el cómputo, separar una tarea en sub-tareas que cooperan ejecutándose paralelamente. Para planificar tareas de manera tal que se pueden ejecutar en paralelo.

Gestión de memoria: en un sistema monoprogramado, la memoria principal se divide en dos partes: una parte para el sistema operativo, monitor residente y núcleo, y otra parte para el programa que se ejecuta en ese instante. En un sistema multiprogramado, la parte de "usuario" de la memoria debe subdividirse aún más para hacer sitio a varios procesos. La tarea de subdivisión la lleva a cabo dinámicamente el sistema operativo y se conoce como gestión de memoria.

En un sistema multiprogramado resulta vital una gestión efectiva de la memoria. Si solo hay unos pocos procesos en memoria, entonces la mayor parte del tiempo estarán esperando a la E/S y el procesador estará desocupado. Por ello, hace falta repartir eficientemente la memoria para meter tantos procesos como sea posible. La división lógica de la memoria para alojar múltiples procesos, debe ser asignada eficientemente para contener el mayor número de procesos como sea posible. Cuanto más procesos estén en memoria, más procesos competirán por la CPU, y tendrá más probabilidad que la CPU no esté ociosa.

Requisitos de la memoria: al realizar un estudio de los diversos mecanismos y políticas relacionadas con la gestión de memoria, vale la pena tener en mente los requisitos que se intentan satisfacer propone cinco requisitos:

- **Reubicación:** en un sistema multiprogramado, la memoria disponible se encuentra normalmente compartida por varios procesos. En general, el programador no puede conocer por adelantado que otros programas residirán en memoria en el momento de la ejecución del programa. Además, se busca poder cargar y descargar los procesos activos en la memoria principal para maximizar el uso del procesador, manteniendo una gran reserva de procesos listos para ejecutar. Una vez que un programa haya sido descargado al disco, se limitará a declarar que, cuando vuelva a ser cargado, debe situarse en la misma región de memoria principal que antes. De este modo, se sabe antes de tiempo donde debe situarse un programa y hay que permitir que el programa pueda moverse en memoria principal como resultado de un intercambio. Esta situación plantea algunos asuntos técnicos relativos al direccionamiento, como se muestra en la figura, que representa la imagen de un proceso. Por simplicidad, se supondrá que la imagen del proceso ocupa una región contigua de la memoria principal. Sin duda, el sistema operativo tiene que conocer la ubicación de la información de control del proceso y de la pila de ejecución, así como el punto de partida para comenzar la ejecución del programa para dicho proceso. Puesto que el sistema operativo gestiona la memoria y es responsable de traer el proceso a memoria principal, estas direcciones deben ser fáciles de conseguir. Además, el procesador debe ocuparse de las referencias a memoria dentro del programa. Las instrucciones de bifurcación deben contener la dirección que haga referencia a la instrucción que se vaya a ejecutar a continuación. Las instrucciones que hagan referencia a datos deben contener la dirección del byte o de la palabra de datos referenciada. De algún modo, el hardware del procesador y el software del sistema operativo deben ser capaces de traducir las referencias a memoria encontradas en el código del programa a las direcciones físicas reales que reflejen la posición actual del programa en memoria principal.
- **Protección:** cada proceso debe protegerse contra interferencias no deseadas de otros procesos, tanto accidentales como intencionados. Así pues, el código de un proceso no puede hacer referencia a posiciones de memoria de otros procesos, con fines de lectura o escritura, sin permiso. Hasta cierto punto, satisfacer las exigencias de reubicación aumenta la dificultad de satisfacción de las exigencias de protección. Puesto que se desconoce la ubicación de un programa en memoria principal, es imposible comprobar las direcciones absolutas durante la compilación para asegurar la protección. Es más, la mayoría de los lenguajes de programación permiten el cálculo dinámico de direcciones durante la ejecución, generando, por ejemplo, un índice de un vector o un puntero a una estructura de datos. Por tanto, todas las referencias a memoria generadas por un proceso deben comprobarse durante la ejecución para asegurar que solo hacen referencia al espacio de memoria destinado a dicho proceso. Afortunadamente, como se verá, los



mecanismos que respaldan la reubicación también forman parte básica del cumplimiento de las necesidades de protección.

La imagen del proceso de la figura ilustra las necesidades de protección. Normalmente, un proceso de usuario no puede acceder a ninguna parte del sistema operativo, tanto programa como datos. De nuevo, el programa de un proceso no puede en general bifurcar hacia una instrucción de otro proceso. Además, sin un acuerdo especial, el programa de un proceso no puede acceder al área de datos de otro proceso. El procesador debe ser capaz de abandonar tales instrucciones en el momento de la ejecución.

Las exigencias de protección de memoria pueden ser satisfechas por el procesador en vez de por el sistema operativo. Esto es debido a que el sistema operativo no puede anticiparse a todas las referencias a memoria que hará un programa. Incluso si tal anticipación fuera posible, sería prohibitivo en términos de tiempo consumido el proteger cada programa por adelantado de posibles violaciones de referencias a memoria. Así pues, solo es posible evaluar la tolerancia de una referencia a memoria durante la ejecución de la instrucción que realiza la referencia. Para llevar esto a cabo, el hardware del procesador debe poseer dicha capacidad.

- **Compartición:** cualquier mecanismo de protección que se implemente debe tener la flexibilidad de permitir el acceso de varios procesos a la misma zona de memoria principal. Por ejemplo, si una serie de procesos están ejecutando el mismo programa, resultaría beneficioso permitir a cada proceso que acceda a la misma copia del programa, en lugar de tener cada uno su propia copia aparte. Los procesos que cooperan en una tarea pueden necesitar acceso compartido a la misma estructura de datos. El sistema de gestión de memoria debe permitir accesos controlados a las áreas compartidas de la memoria, sin comprometer la protección básica.
- **Organización lógica:** de forma casi invariable, la memoria principal de un sistema informático se organiza como un espacio de direcciones lineal o unidimensional que consta de una secuencia de bytes o palabras. La memoria secundaria se organiza de forma similar. Si bien esta organización refleja fielmente el hardware de la máquina, no se corresponde con la forma en la que los programas están construidos habitualmente. La mayoría de los programas se organizan en módulos, algunos de los cuales no son modificables y otros contienen datos que se pueden modificar. Si el sistema operativo y el hardware del computador pueden tratar de forma efectiva los programas de usuario y los datos en forma de módulos de algún tipo, se conseguirá una serie de ventajas, tales como:
 - I. Los módulos pueden escribirse y compilarse independientemente, mientras que el sistema resuelve durante la ejecución todas las referencias de un módulo a otro.
 - II. Pueden otorgarse varios grados de protección a los distintos módulos.
 - III. Es posible introducir mecanismos por medio de los cuales los procesos puedan compartir módulos. La ventaja de ofrecer compartición a nivel de modulo es que esto se corresponde con la visión del problema que tiene el usuario y es fácil para el usuario especificar la compartición que desea.
- **Organización física:** la memoria del computador se organiza en, al menos, dos niveles: memoria principal y memoria secundaria. La memoria principal ofrece un acceso rápido con un coste relativamente alto. La memoria principal es volátil, no proporciona almacenamiento permanente. La memoria secundaria es más lenta y barata que la memoria principal y, normalmente, no es volátil. De este modo, una memoria secundaria de gran capacidad puede permitir un almacenamiento a largo plazo de programas y datos, al tiempo que una memoria principal pequeña mantiene los programas y datos de uso actual.

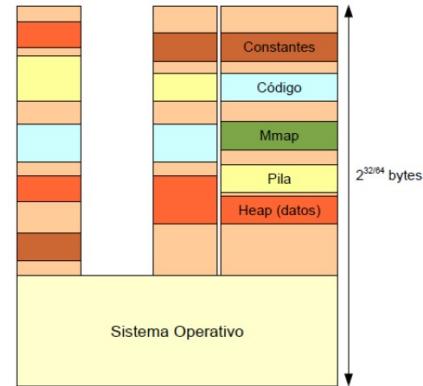
La organización del flujo de información entre la memoria principal y la secundaria tiene un gran interés en el sistema. La responsabilidad de este flujo podría asignarse al programador, pero esto es impracticable e indeseable, debido a dos razones:

- I. La memoria principal disponible para un programa y sus datos puede ser insuficiente. El programador debe emplear una práctica que se conoce como superposición (overlaying), en la cual el programa y los datos se organizan de tal forma que puede haber varios módulos asignados a la misma región de memoria, con un programa principal responsable del intercambio de lo módulo según se necesite. Incluso con la ayuda de herramientas de compilación, la programación superpuesta malgasta el tiempo del programador.
- II. En un entorno multiprogramado, el programador no conoce durante la codificación cuánto espacio habrá disponible o donde estará este espacio.

Espacio de direcciones: la memoria principal y los registros integrados dentro del propio procesador son las únicas áreas de almacenamiento a las que

la CPU puede acceder directamente. Hay instrucciones de máquina que toman como argumentos direcciones de memoria, pero no existe ninguna instrucción que acepte direcciones de disco. Por tanto, todas las instrucciones en ejecución y los datos utilizados por esas instrucciones deberán encontrarse almacenados en uno de esos dispositivos de almacenamiento de acceso directo. Si los datos no se encuentran en memoria, deberán llevarse hasta allá antes de que la CPU pueda operar con ellos.

El rango de direcciones a memoria posibles que un proceso puede utilizar para dirigir sus instrucciones y datos, depende del tamaño de la memoria y de la arquitectura que se utiliza, 32 bits o 64 bits. La ubicación real del proceso en memoria debe ser independiente del espacio asignado para guardar sus datos.



Espacio de direcciones lógico y físico: una dirección generada por la CPU se denomina comúnmente dirección lógica, mientras que una dirección vista por la unidad de memoria se denomina comúnmente dirección física.

Los métodos de reasignación en tiempo de compilación y en tiempo de carga generan direcciones lógicas y físicas idénticas. Sin embargo, el esquema de reasignación de direcciones en tiempo de ejecución hace que las direcciones lógica y física difieran. En este caso, usualmente decimos que la dirección lógica es una dirección virtual. El conjunto de todas las direcciones lógicas generadas por un programa es lo que se denomina un espacio de direcciones lógicas; el conjunto de todas las direcciones físicas correspondientes a estas direcciones lógicas es un espacio de direcciones físicas.

La correspondencia entre direcciones virtuales y físicas en tiempo de ejecución es establecida por un dispositivo hardware que se denomina unidad de gestión de memoria, MMU Memory management unit. El registro base se denominará ahora registro de reubicación. El valor contenido en el registro de reubicación suma a todas las direcciones generadas por un proceso de usuario en el momento de enviarlas a memoria. El programa de usuario maneja direcciones *lógicas* y el hardware de conversión, mapeo, de memoria convierte esas direcciones lógicas en direcciones físicas. La ubicación final de una dirección de memoria referenciada no se determina hasta que se realiza esa referencia.

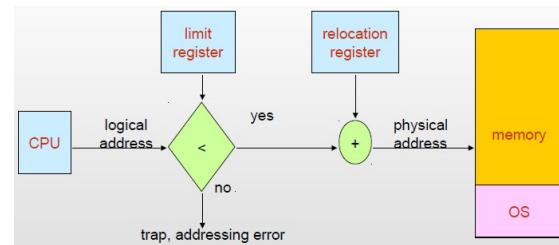
Conversión de direcciones: tenemos que asegurarnos de que cada proceso disponga de un espacio de memoria separado. Para hacer esto, debemos poder determinar el rango de direcciones legales a las que el proceso pueda acceder y garantizar también que el proceso sólo acceda a esas direcciones legales. Podemos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

La protección del espacio de memoria se consigue haciendo que el hardware de la CPU compare todas las direcciones generadas en modo usuario con el contenido de esos registros. Cualquier intento, por parte de un programa que se esté ejecutando en modo usuario, de acceder a la memoria del sistema operativo o a la memoria de otros usuarios hará que se produzca una interrupción hacia el sistema operativo, que tratará dicho intento como un error fatal. Este esquema evita que un programa de usuario modifique el código y las estructuras de datos del sistema operativo o de otros usuarios.

Los registros base y límite sólo pueden ser cargados por el sistema operativo, que utiliza una instrucción privilegiada especial. Puesto que las instrucciones privilegiadas sólo pueden ser ejecutadas en modo kernel y como sólo el sistema operativo se ejecuta en modo kernel, únicamente el sistema operativo podrá cargar los registros base y límite. Este esquema permite al sistema operativo modificar el valor de los registros, pero evita que los programas de usuario cambien el contenido de esos registros.

Podemos proporcionar estas características utilizando un registro de reubicación, con un registro límite. El registro de reubicación contiene el valor de la dirección física más pequeña, mientras que el registro límite contiene el rango de las direcciones lógicas. Con los registros de reubicación y de límite, cada dirección lógica debe ser inferior al valor contenido en el registro límite; la MMU convertirá la dirección lógica dinámicamente sumándole el valor contenido en el registro de reubicación. Luego su valor se envía a través del context switch.

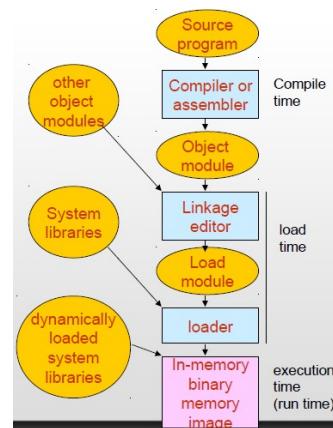
Cuando el planificador de la CPU selecciona un proceso para su ejecución, el despachador carga en los registros de reubicación y de límite los valores correctos, como parte del proceso de cambio de contexto. Puesto que todas las direcciones generadas por la CPU se comparan con estos registros, este mecanismo nos permite proteger tanto al sistema operativo como a los programas y datos de los otros usuarios de las posibles modificaciones que pudiera realizar este proceso en ejecución.



Binding de direcciones: las direcciones del programa fuente son generalmente simbólicas. Normalmente, un compilador se encargará de reasignar estas direcciones simbólicas al direcciones reubicables. El editor de montaje o cargador se encargará, a su vez, de reasignar las direcciones reubicables a direcciones absolutas. Cada operación de reasignación constituye una relación de un espacio de direcciones a otro.

Clásicamente, la reasignación de las instrucciones y los datos a direcciones de memoria puede realizarse en cualquiera de los pasos:

- I. **Tiempo de compilación:** si sabemos en el momento de realizar la compilación dónde va a residir el proceso en memoria, podremos generar código absoluto. Por ejemplo,



si sabemos que un proceso de usuario va a residir en una zona de memoria que comienza en la ubicación R , el código generado por el compilador comenzará en dicha ubicación y se extenderá a partir de ahí. Si la ubicación inicial cambiase en algún instante posterior, entonces sería necesario recompilar ese código.

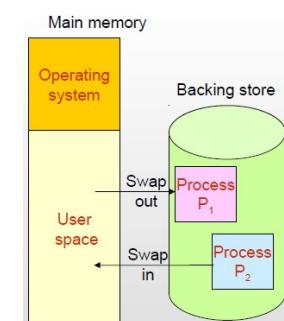
- II. **Tiempo de carga:** si no conocemos en tiempo de compilación dónde va a residir el proceso en memoria, el compilador deberá generar código reubicable. En este caso, se retarda la reasignación final hasta el momento de la carga. Si cambia la dirección inicial, tan sólo es necesario volver a cargar el código de usuario para incorporar el valor modificado.
- III. **Tiempo de ejecución:** si el proceso puede desplazarse durante su ejecución desde un segmento de memoria a otro, entonces es necesario retardar la reasignación hasta el instante de la ejecución. La reubicación se puede realizar fácilmente. El mapeo entre virtuales y físicas es realizado por hardware, dicho hardware se denomina MMU, Memory Management Unit.

Swapping: un proceso debe estar en memoria para ser ejecutado. Sin embargo, los procesos pueden ser intercambiados temporalmente, sacándolos de la memoria y almacenándolos en un almacén de respaldo y volviéndolos a llevar luego a memoria para continuar su ejecución. Por ejemplo, suponga que estamos utilizando un entorno de multiprogramación con un algoritmo de planificación de CPU basado en turnos. Cuando termina un cuento de tiempo, el gestor de memoria comienza a sacar, swapped out, de ésta el proceso que acaba de terminar y a cargar en el espacio de memoria liberado por otro proceso. Mientras tanto, el planificador de la CPU asignará un cuento de tiempo a algún otro proceso que ya se encuentre en memoria. Cada vez que un proceso termine su cuento asignado, se intercambiará por otro proceso. Idealmente, el gestor de memoria puede intercambiar los procesos con la suficiente rapidez como para que haya siempre algunos procesos en memoria, listos para ejecutarse, cuando el planificador de la CPU quiera asignar el procesador a otra tarea.

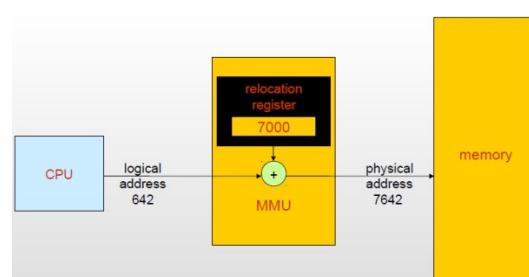
Para los algoritmos de planificación con prioridad se utiliza una variante de esta política de swapping, intercambio. Si llega un proceso de mayor prioridad y ese proceso desea ser servido, el gestor de memoria puede descargar el proceso de menor prioridad y, a continuación, cargar y ejecutar el que tiene una prioridad mayor. Cuando termine el proceso de mayor prioridad, puede intercambiarse por el proceso de menor prioridad, que podrá entonces continuar su ejecución. Normalmente, un proceso descargado se volverá a cargar en el mismo espacio de memoria que ocupaba anteriormente. Esta restricción está dictada por el método de reasignación de las direcciones. Si la reasignación se realiza en tiempo de ensamblado o en tiempo de carga, entonces no resulta sencillo mover el proceso a una ubicación diferente. Sin embargo, si se está utilizando reasignación en tiempo de ejecución sí que puede moverse el proceso a un espacio de memoria distinto, porque las direcciones físicas se calculan en tiempo de ejecución.

Un proceso puede ser temporalmente sacado de la memoria, swapped out, a un disco de manera de permitir la ejecución de procesos.

Si se descarga considerando las direcciones físicas, al hacer swapped in se debe cargar en el mismo espacio de memoria que ocupaba antes; y si se descarga considerando las direcciones lógicas, al hacer swapped in se puede cargar en cualquier espacio de direcciones de memoria.



MMU, Memory Management Unit: la correspondencia entre direcciones virtuales y



físicas en tiempo de ejecución es establecida por un dispositivo hardware que se denomina unidad de gestión de memoria, MMU. El registro base se denominará ahora registro de reubicación. El valor contenido en el registro de reubicación suma a todas las direcciones generadas por un proceso de usuario en el momento de enviarlas a memoria. Por ejemplo, si la base se encuentra en la dirección 14000, cualquier intento del usuario de direccionar la posición de memoria cero se reubicará dinámicamente en la dirección 14000; un acceso a la ubicación 346 se convertirá en la ubicación 14346. El programa de usuario nunca ve las direcciones reales y los procesos nunca usan direcciones físicas.

Asignación de memoria: la memoria principal debe albergar tanto el sistema operativo como los diversos procesos del usuario. Por tanto, necesitamos asignar las distintas partes de la memoria principal de la forma más eficiente posible.

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

La realización de asignación de memoria se puede realizar dependiendo de las diferentes particiones que se tienen en el sistema:

- **Única partición:** los procesos ocupan una única partición de memoria y la protección se implementa a través de un registro límite y un registro de reubicación, como vimos antes.
- **Particiones múltiples:** cada partición puede contener exactamente un proceso, de modo que el grado de multiprogramación estará limitado por el número de particiones disponibles, cuando una partición está libre, se selecciona un proceso de la cola de entrada y se lo carga en dicha partición. Cuando el proceso termina, la partición pasa a estar disponible para otro proceso. Las técnicas que cumplen con estas particiones son fijas o dinámicas.

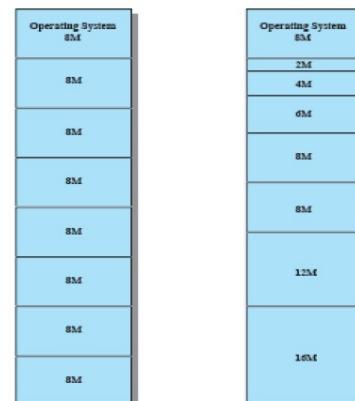
Técnica de partición fija: en la mayoría de los esquemas de gestión de memoria, se puede suponer que el sistema operativo ocupa una parte fija de memoria principal y que el resto de la memoria está disponible para ser usado por varios procesos. El esquema más sencillo de gestión de la memoria disponible es dividirla en regiones con límites fijos.

Dos alternativas de partición fija son particiones de igual tamaño o de distinto tamaño.

- **Particiones de igual tamaño:** cualquier proceso cuyo tamaño sea menor o igual que el tamaño de la partición puede cargarse en cualquier partición libre. Si todas las particiones están ocupadas y no hay procesos residentes en estado listo o ejecutando, el sistema operativo puede sacar un proceso de alguna de las particiones y cargar otro proceso de forma que haya trabajo para el procesador.

Estas particiones presentan dos dificultades:

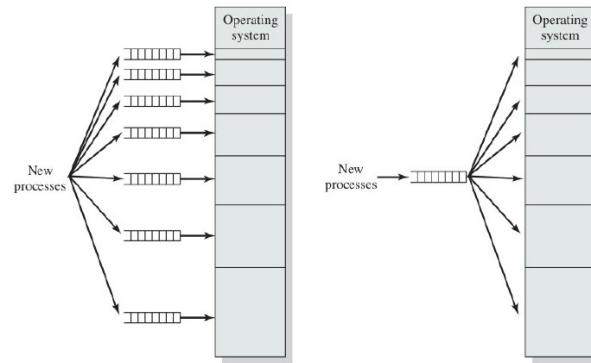
- I. Un programa puede ser demasiado grande para caber en la partición. En este caso, el programador debe diseñar el programa mediante superposiciones, para que solo una parte del programa esté en memoria principal en cada instante. Cuando se necesita un módulo que no está presente, el programa de usuario debe cargar dicho módulo en la partición del programa, superponiéndose a los programas y datos que se encuentren en ella.
- II. El uso de memoria principal es extremadamente ineficiente. Cualquier programa, sin importar lo pequeño que sea, ocupará una partición



completa. En el ejemplo, puede haber un programa que ocupe menos de 128Kb de memoria y, aun así, ocuparía una partición de 512Kb cada vez que se cargase. Este fenómeno, en el que se malgasta el espacio interno de una partición cuando el bloque de datos cargado sea más pequeño que la partición, se denomina fragmentación interna.

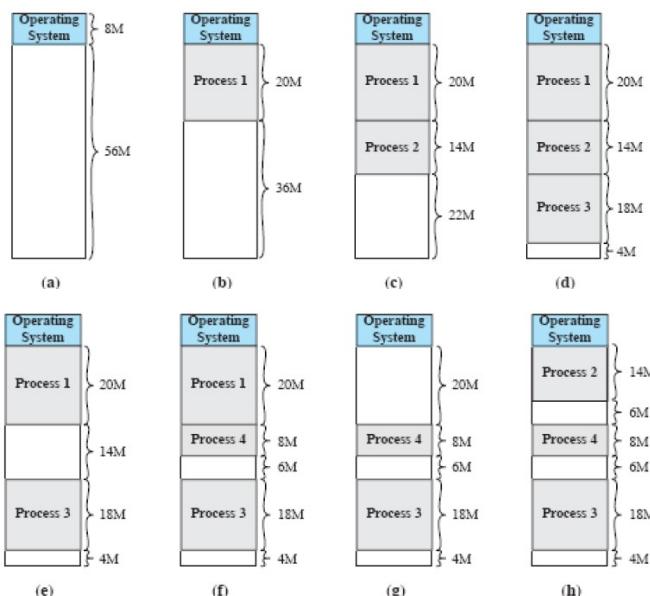
- **Particiones de diferentes tamaños:** tiene regiones definidas con límites fijados, estas particiones evita el problema de la particiones de igual tamaño, ya que los procesos pequeños pueden usar particiones pequeñas y se puede prever que algunas particiones grandes para procesos grandes. El problema que presenta es la complejidad en el algoritmo de selección de partición para un proceso.

Los algoritmos de ubicación difieren dependiendo si es de tamaños fijos y diferentes. Con particiones del mismo tamaño, la ubicación de un proceso en memoria es trivial. Mientras haya alguna partición libre, puede cargarse un proceso en esa partición. Puesto que todas las particiones son de igual tamaño, no importa la partición que se use. Si todas las particiones están ocupadas con procesos que no están listos para ejecutar, uno de esos procesos debe sacarse y hacer sitio para un nuevo proceso.



Con particiones de distintos tamaños, hay dos maneras posibles de asignar los procesos a las particiones. La forma más simple es asignar cada proceso a la partición más pequeña en la que quepa. En este caso, hace falta una cola de planificación para cada partición, que alberge los procesos expulsados cuyo destinatario es dicha partición, figura a. La ventaja de este enfoque es que los procesos están siempre asignados de forma que se minimiza la memoria desperdiciada dentro de cada partición. Sin embargo, aunque esta técnica parece optima desde el punto de vista de una partición individual, no lo es desde el punto de vista del sistema global. Por ejemplo, donde no hay procesos con un tamaño comprendido entre 768K y 1M en un determinado instante. En este caso, la partición de 768K permanecerá sin usar, incluso aunque algún proceso más pequeño pudiera haber sido asignado a la misma.

Técnica de particiones dinámicas: las particiones son variables en número y longitud. Cuando se trae un proceso a memoria principal, se le asigna exactamente tanta memoria como necesita y no más. En la figura se muestra un ejemplo que usa 1MB de memoria principal. Al principio, la memoria principal está vacía, exceptuando el sistema operativo, caso a. Se cargan los tres primeros procesos, empezando en donde acaba el sistema operativo y ocupando solo un espacio suficiente para cada proceso, casos b, c y d. Esto deja un



"hueco" al final de la memoria demasiado pequeño para un cuarto proceso. En algún momento, ninguno de los procesos en memoria está listo. Así pues, el sistema operativo saca al proceso 2, caso e, que deja sitio suficiente para cargar un nuevo proceso, el proceso 4, caso f. Puesto que el proceso 4 es más pequeño que el proceso 2, se crea otro hueco pequeño. Más tarde, se alcanza un punto en el que ninguno de los procesos que están en memoria principal está listo y el proceso 2, que está en estado listo, pero suspendido, está disponible. Puesto que no hay suficiente sitio en memoria para el proceso 2, el sistema operativo expulsa al proceso 1, caso g, y carga de nuevo el proceso 2, caso h.

Este método comienza bien, pero, finalmente, desemboca en una situación en la que hay un gran número de huecos pequeños en memoria. Conforme pasa el tiempo, la memoria comienza a estar más fragmentada y su rendimiento decae. Este fenómeno se denomina fragmentación externa y se refiere al hecho de que la memoria externa a todas las particiones se fragmenta cada vez más, a diferencia de la fragmentación interna.

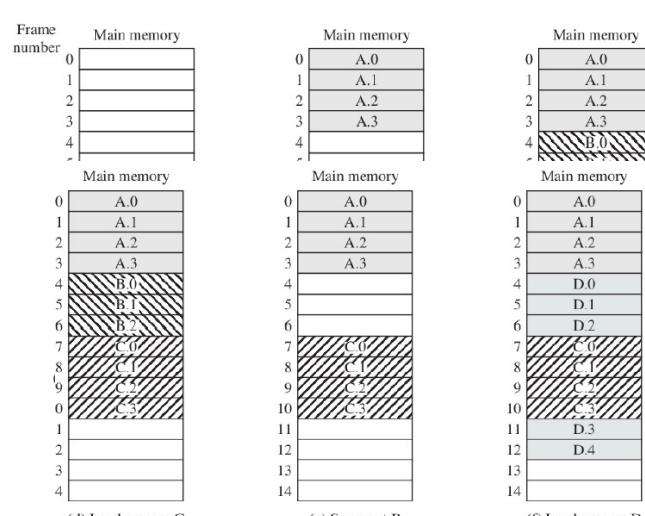
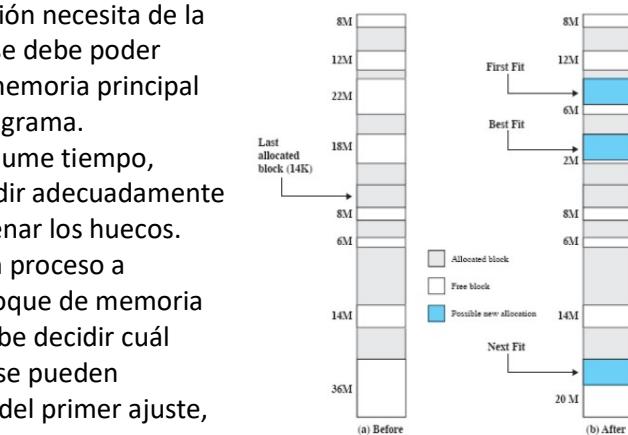
Una técnica para superar la fragmentación externa es la compactación: De vez en cuando, el sistema operativo desplaza los procesos para que estén contiguos de forma que toda la memoria libre quede junta en un bloque. La compactación necesita de la capacidad de reubicación dinámica. Es decir, se debe poder mover un programa de una región a otra de memoria principal sin invalidar las referencias a memoria del programa.

Puesto que la compactación de memoria consume tiempo, atañe al diseñador del sistema operativo decidir adecuadamente como asignar un proceso a memoria, como llenar los huecos.

Cuando llega el momento de cargar o traer un proceso a memoria principal y, si hay libre más de un bloque de memoria de tamaño suficiente, el sistema operativo debe decidir cuál asignar. Los tres algoritmos de ubicación que se pueden considerar son el del mejor ajuste, best-fit, el del primer ajuste, first-fit y el del siguiente ajuste, next fit.

- Best fit:** selecciona la partición más pequeña que contiene al proceso, realiza mucho overhead en la búsqueda. En particiones dinámicas se generan muchos huecos pequeños de memoria libre, fragmentación interna.
- First fit:** recorre las particiones libres en orden buscando la primera que pueda alojar al proceso, es el algoritmo más sencillo y suele ser mejor y más rápido que los demás.
- Next fit:** mantiene las particiones libres como una lista circular. Funciona como el First Fit solo que empieza desde la posición actual en la lista y no desde el principio. Este selecciona la primera partición que encuentra que pueda contener al proceso. Tiende a producir resultados ligeramente peores que el First Fit, pero utiliza más rápidamente particiones grandes que puedan existir al final de la lista.

Paginación: tanto las particiones de tamaño fijo como las de tamaño variable hacen un uso ineficiente de la memoria; las primeras generan fragmentación interna, mientras que las segundas originan fragmentación externa. Supóngase que la memoria principal se encuentra particionada en trozos iguales de tamaño fijo relativamente pequeños y que cada proceso está dividido



también en pequeños trozos de tamaño fijo y del mismo tamaño que los de memoria. En tal caso, los trozos del proceso, conocidos como páginas, pueden asignarse a los trozos libres de memoria, conocidos como marcos o marcos de página. Se verá que el espacio malgastado en memoria para cada proceso por fragmentación interna consta solo de una fracción de la última página del proceso. Además, no hay fragmentación externa.

En la figura se muestra un ejemplo del uso de páginas y marcos. En un instante dado, algunos de los marcos de memoria están en uso y otros están libres. El sistema operativo mantiene una lista de los marcos libres. El proceso A, almacenado en disco, consta de cuatro páginas. Cuando llega el momento de cargar este proceso, el sistema operativo busca cuatro marcos libres y carga las cuatro páginas del proceso A en los cuatro marcos, figura b. El proceso B, que consta de tres páginas y el proceso C, que consta de cuatro, se cargan a continuación. Más tarde, el proceso B se suspende y es expulsado de memoria principal. Despues, todos los procesos de memoria principal están bloqueados y el sistema operativo tiene que traer un nuevo proceso, el proceso D, que consta de cinco páginas.

Esto no impide al sistema operativo cargar D, puesto que se puede emplear de nuevo el concepto de dirección lógica. Ya no será suficiente con un simple registro base. En su lugar, el sistema operativo mantiene una tabla de páginas para cada proceso. La tabla de páginas muestra la posición del marco de cada página del proceso. Dentro del programa, cada dirección lógica constará de un número de página y de un desplazamiento dentro de la página. Hay que recordar que, en el caso de la partición simple, una dirección lógica era la posición de una palabra relativa al comienzo del programa; el procesador realizaba la traducción a dirección física. Con paginación, el hardware del procesador también realiza la traducción de direcciones lógicas a físicas. Ahora, el procesador debe saber cómo acceder a la tabla de páginas del proceso actual. Dada una dirección lógica, número de página desplazamiento, el procesador emplea la tabla de páginas para obtener una dirección física, numero de marco, desplazamiento.

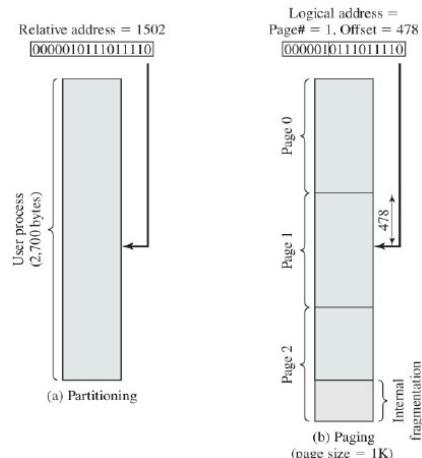
Continuando con el ejemplo, las cinco páginas del proceso D se cargan en los marcos 4, 5, 6, 11 y 12. La figura muestra las distintas tablas de páginas en este instante. Cada tabla de páginas contiene una entrada por cada página

<table border="1"> <tr><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td></tr> <tr><td>2</td><td>2</td></tr> <tr><td>3</td><td>3</td></tr> </table>	0	0	1	1	2	2	3	3	<table border="1"> <tr><td>0</td><td>—</td></tr> <tr><td>1</td><td>—</td></tr> <tr><td>2</td><td>—</td></tr> </table>	0	—	1	—	2	—	<table border="1"> <tr><td>0</td><td>7</td></tr> <tr><td>1</td><td>8</td></tr> <tr><td>2</td><td>9</td></tr> <tr><td>3</td><td>10</td></tr> </table>	0	7	1	8	2	9	3	10	<table border="1"> <tr><td>0</td><td>4</td></tr> <tr><td>1</td><td>5</td></tr> <tr><td>2</td><td>6</td></tr> <tr><td>3</td><td>11</td></tr> <tr><td>4</td><td>12</td></tr> </table>	0	4	1	5	2	6	3	11	4	12	Free frame list
0	0																																			
1	1																																			
2	2																																			
3	3																																			
0	—																																			
1	—																																			
2	—																																			
0	7																																			
1	8																																			
2	9																																			
3	10																																			
0	4																																			
1	5																																			
2	6																																			
3	11																																			
4	12																																			
Process A page table	Process B page table	Process C page table	Process D page table																																	

del proceso, por lo que la tabla se indexa fácilmente por número de página, comenzando en la página 0. En cada entrada de la tabla de páginas se encuentra el número de marco en memoria, si lo hay, que alberga la página correspondiente. Además, el sistema operativo mantiene una lista de marcos libres con todos los marcos de memoria que actualmente están vacíos y disponibles para las páginas.

Así pues, se puede comprobar que la paginación simple, tal y como se describe, es similar a la partición estática. Las diferencias están en que, con paginación, las particiones son algo más pequeñas, un programa puede ocupar más de una partición y estas no tienen por qué estar contiguas.

Paginación-direcciones lógicas: para aplicar convenientemente este esquema de paginación, el tamaño de la página y, por tanto, el tamaño del marco, deben ser una potencia de 2. En este caso, la dirección relativa, definida en relación al origen del programa y la dirección lógica, expresada como un número de página y un desplazamiento, son las mismas. En la figura se muestra un ejemplo, donde se



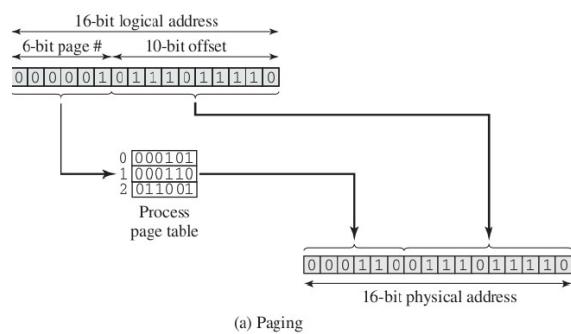
emplean direcciones de 16 bits y el tamaño de página es de $1K = 1024$ bytes. La dirección relativa 1502 es 0000010111011110 en binario. Con un tamaño de página de 1K, se necesitan 10 bits para el campo de desplazamiento, dejando 6 bits para el número de página. De este modo, un programa puede estar formado por un máximo de $2^6 = 64$ páginas de 1Kb cada una. Como muestra la figura b, la dirección relativa 1502 se corresponde con un desplazamiento de 478, 0111011110, en la página 1, 000001, lo que genera el mismo número de 16 bits, 0000010111011110.

Dos son las consecuencias de usar un tamaño de página potencia de dos. Primero, el esquema de direccionamiento lógico es transparente al programador, al ensamblador y al montador. Cada dirección lógica de un programa, número de página, desplazamiento es idéntica a su dirección relativa. Segundo, resulta relativamente sencillo realizar una función hardware para llevar a cabo la traducción de direcciones dinámicas durante la ejecución. Considérese una dirección de "n+m" bits, en la que los n bits más significativos son el número de página y los m bits menos significativos son el desplazamiento. En el ejemplo, figura b, n=6 y m=10. Para la traducción de direcciones hay que dar los siguientes pasos:

- Obtener el número de página de los n bits más significativos de la dirección lógica.
- Emplear el número de página como índice en la tabla de páginas del proceso para encontrar el número de marco k.
- El comienzo de la dirección física del marco es $k \times 2^M$ y la dirección física del byte referenciado en este número más el desplazamiento.

En el ejemplo, se tiene la dirección lógica 0000010111011110, que se corresponde con el número de página 1 y desplazamiento 478. Suponiendo que esta página reside en el marco de memoria principal 6 = 000110. Entonces la dirección física es el marco 6, desplazamiento 478 = 0001100111011110, figura a.

Resumiendo, mediante la paginación simple, la memoria principal se divide en pequeños marcos del mismo tamaño. Cada proceso se divide en páginas del tamaño del marco; los procesos pequeños necesitarán pocas páginas, mientras que los procesos grandes necesitarán más. Cuando se introduce un proceso en memoria, se cargan todas sus páginas en los marcos libres y se rellena su tabla de páginas. Esta técnica resuelve la mayoría de los problemas inherentes a la partición.

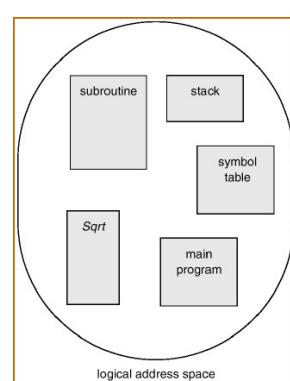


Segmentación: es un esquema de gestión de memoria que soporta esta visión de la memoria que tienen los usuarios. Un espacio lógico de direcciones es una colección de segmentos y cada segmento tiene un nombre y una longitud. Las direcciones especifican tanto el nombre del segmento como el desplazamiento dentro de ese segmento. El usuario especifica cada dirección proporcionando dos valores: un nombre de segmento y un desplazamiento.

Por simplicidad de implementación, los segmentos están numerados y se hacen referencia a ellos mediante un número de segmento, en lugar de utilizar un nombre de segmento. Así, una dirección lógica estará compuesta por una pareja del tipo: <número-segmento, desplazamiento>.

Normalmente, el programa del usuario se compila y el compilador construye automáticamente los segmentos para reflejar el programa de entrada. Se pueden crear segmentos separados para:

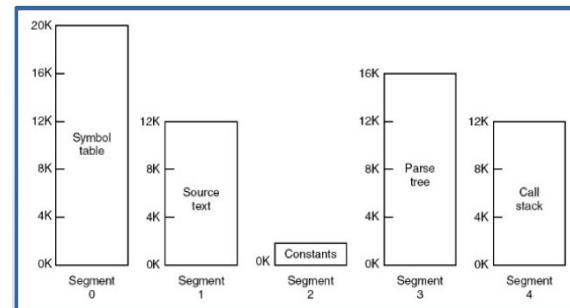
- El código.
- Las variables globales.
- El cúmulo de memoria a partir del cual se asigna la memoria.



- Las pilas utilizadas por cada hebra de ejecución.

Como consecuencia del empleo de segmentos de distinto tamaño, la segmentación resulta similar a la partición dinámica. En ausencia de un esquema de superposición o del uso de memoria virtual, sería necesario cargar en memoria todos los segmentos de un programa para su ejecución. La diferencia, en comparación con la partición dinámica, radica en que, con segmentación, un programa puede ocupar más de una partición y estas no tienen por qué estar contiguas. La segmentación elimina la fragmentación interna, pero, como la partición dinámica, sufre de fragmentación externa. Sin embargo, debido a que los procesos se dividen en un conjunto de partes más pequeñas, la fragmentación externa será menor.

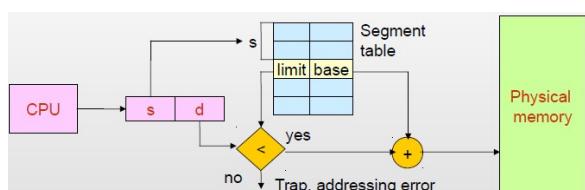
Debido a que cada segmento constituye un espacio de direcciones separado, los distintos segmentos pueden crecer o reducirse de manera independiente, sin afectar unos a otros. Si una pila en cierto segmento necesita más espacio de direcciones para crecer, puede tenerlo, ya que no hay nada más en su espacio de direcciones con lo que se pueda topar. Desde luego que un segmento se puede llenar, pero por lo general los segmentos son muy grandes, por lo que esta ocurrencia es rara. Para especificar una dirección en esta memoria segmentada o bidimensional, el programa debe suministrar una dirección en dos partes, un número de segmento y una dirección dentro del segmento. La figura ilustra el uso de una memoria segmentada para las tablas del compilador que vimos antes.



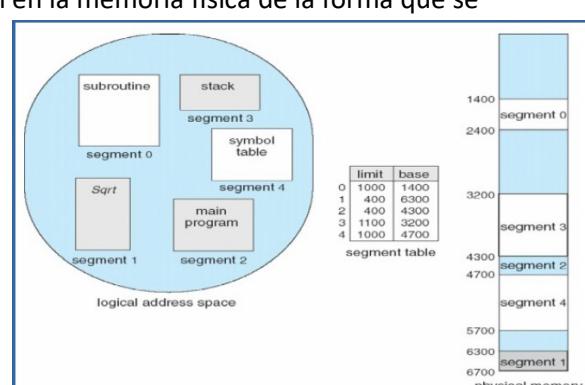
Aunque el usuario puede ahora hacer referencia a los objetos del programa utilizando una dirección tridimensional, la memoria física real continúa siendo, por supuesto, una secuencia unidimensional de bytes. Por tanto, deberemos definir una implementación para mapear las direcciones bidimensionales definidas por el usuario sobre las direcciones físicas unidimensionales. Este mapeo se lleva a cabo mediante una tabla de segmentos. Cada entrada de la tabla de segmentos tiene una dirección base del segmento y un límite del segmento. La dirección base del segmento contiene la dirección física inicial del lugar donde el segmento reside dentro de la memoria, mientras que el límite del segmento especifica la longitud de éste.

El uso de una tabla de segmentos se ilustra en la figura. Una dirección lógica estará compuesta de dos partes: un número de segmento, s , y un desplazamiento dentro de ese segmento, d . El número de segmento se utiliza como índice para la tabla de segmentos. El desplazamiento d de la dirección lógica debe estar comprendido entre 0 y el límite del segmento; si no lo está, se producirá una interrupción hacia el sistema operativo, intento de direccionamiento lógico más allá del final del segmento. Cuando un desplazamiento es legal, se lo suma a la dirección base del segmento para generar la dirección de memoria física del byte deseado. La tabla de segmentos es, por tanto, esencialmente una matriz de parejas de registros base-límite.

Como ejemplo, considere la situación mostrada en la Figura 8.20. Tenemos cinco segmentos, numerados de 0 a 4. Los segmentos se almacenan en la memoria física de la forma que se



muestra. La tabla de segmentos dispone de una tabla separada para cada segmento, en la que se indica la dirección inicial del segmento en la memoria física, la base, y la longitud de ese segmento, el límite. Por ejemplo, el segmento 2



tiene 400 bytes de longitud y comienza en la ubicación 4300. Por lo tanto, una referencia al byte 53 del segmento 2 se corresponderá con la posición $4300 + 53 = 4353$. Una referencia al segmento 3, byte 852, se corresponderá con la posición 3200, la base del segmento 3, $+ 852 = 4052$. Una referencia al byte 1222 del segmento 0 provocaría una interrupción hacia el sistema operativo, ya que este segmento sólo tiene 1000 bytes de longitud.

Mientras que la paginación es transparente al programador, la segmentación es generalmente visible y se proporciona como una comodidad para la organización de los programas y datos. Normalmente, el programador o el compilador asignan los programas y los datos a diferentes segmentos. El programa o los datos pueden ser divididos de nuevo en diferentes segmentos. El principal inconveniente de este servicio es que el programador debe ser consciente de la limitación de tamaño máximo de los segmentos.

Otra consecuencia del tamaño desigual de los segmentos es que no hay una correspondencia simple entre las direcciones lógicas y las direcciones físicas. De forma análoga a la paginación, un esquema de segmentación simple hará uso de una tabla de segmentos para cada proceso y una lista de bloques libres en memoria principal. Cada entrada de tabla de segmentos tendrá que contener la dirección de comienzo del segmento correspondiente en memoria principal. La entrada deberá proporcionar también la longitud del segmento para asegurar que no se usan direcciones no válidas. Cuando un proceso pasa al estado running, se carga la dirección de su tabla de segmentos en un registro especial del hardware de gestión de memoria. Considerando una dirección de " $n+m$ " bits, en la que los n bits más significativos son el número de segmento y los m bits menos significativos son el desplazamiento. En el ejemplo anterior, figura c, $n=4$ y $m=12$. Así pues, el máximo tamaño de segmento es $2^{12} = 4096$. Para la traducción de direcciones hay que dar los siguientes pasos:

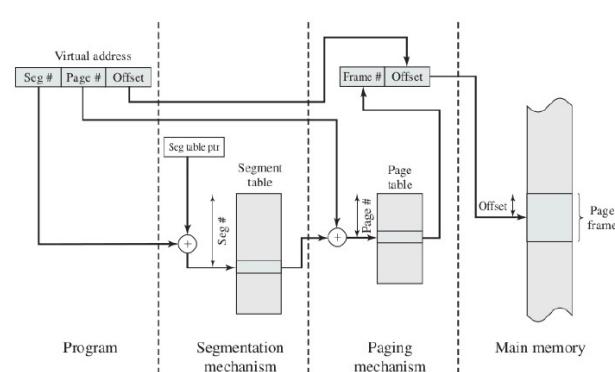
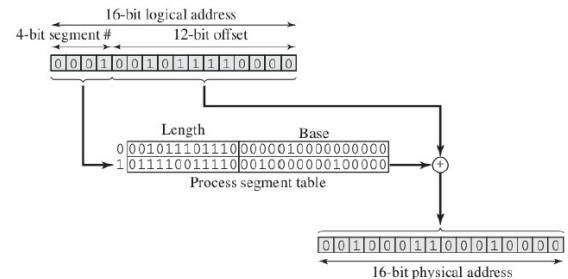
- Extraer el número de segmento de los n bits más significativos de la dirección lógica.
- Emplear el número de segmento como índice en la tabla de segmentos del proceso para encontrar la dirección física de comienzo del segmento.
- Comparar el desplazamiento, expresado por los m bits menos significativos, con la longitud del segmento. Si el desplazamiento es mayor que la longitud, la dirección no es válida.
- La dirección física buscada es la suma de la dirección física de comienzo del segmento más el desplazamiento.

En el ejemplo, se emplea la dirección lógica 0001001011110000, que se corresponde con el número de segmento 1 y desplazamiento 752.

Suponiendo que dicho segmento está en memoria principal y comienza en la dirección física 001000000100000. Entonces la dirección física será 001000000100000 + 001011110000 = 0010001100010000.

En definitiva, con segmentación simple, un proceso se divide en varios segmentos que no tienen por qué ser del mismo tamaño. Cuando un proceso se introduce en memoria, se cargan todos sus segmentos en regiones de memoria libres y se rellena la tabla de segmentos.

Paginación segmentada: paginación y segmentación son técnicas diferentes. Para la segmentación se necesita que estén cargadas en memoria, áreas de tamaños



variables. Si se requiere cargar un segmento en memoria; que antes estuvo en ella y fue removido a memoria secundaria; se necesita encontrar una región de la memoria lo suficientemente grande para contenerlo, lo cual no es siempre factible en cambio "recargar" una página implica solo encontrar un merco de página disponible.

A nivel de paginación, si quiere referenciar en forma cíclicas n páginas, estas deberán ser cargadas una a una generándose varias interrupciones por fallas de páginas; bajo segmentación, esta página podría conformar un solo segmento, ocurriendo una sola interrupción, por falla de segmento. No obstante, si bajo segmentación, se desea acceder un área muy pequeña dentro de un segmento muy grande, este deberá cargarse completamente en memoria, desperdi ciéndose memoria; bajo paginación solo se cargara la página que contiene los ítems referenciados.

Puede hacerse una combinación de segmentación y paginación para obtener las ventajas de ambas. En lugar de tratar un segmento como una unidad contigua, este puede dividirse en páginas. Cada segmento puede ser descrito por su propia tabla de páginas.

Los segmentos son usualmente múltiples de páginas en tamaño, y no es necesario que todas las páginas se encuentren en memoria principal a la vez; además las páginas de un mismo segmento, aunque se encuentren contiguas en memoria virtual; no necesitan estarlo en memoria real.

Las direcciones tienen tres componentes: (s, p, d), donde la primera indica el número del segmento, la segunda el número de la página dentro del segmento y la tercera el desplazamiento dentro de la página. Se deberán usar varias tablas:

- **SMT, tabla de mapas de segmentos:** una para cada proceso. En cada entrada de la SMT se almacena la información descrita bajo segmentación pura, pero en el campo de dirección se indicara la dirección de la PMT, tabla de mapas de páginas, que describe a las diferentes páginas de cada segmento.
- **PMT, tabla de mapas de páginas:** una por segmento; cada entrada de la PMT describe una página de un segmento; en la forma que se presentó la página pura.
- **TBM, tabla de bloques de memoria:** para controlar asignación de páginas por parte del sistema operativo.
- **JT, tabla de Jobs:** que contiene las direcciones de comienzo de cada una de las SMT de los procesos que se ejecutan en memoria.

En el caso, de que un segmento sea de tamaño inferior o igual al de una página, no se necesita tener la correspondiente PMT, actuándose en igual forma que bajo segmentación pura; puede arreglarse un bit adicional, s, a cada entrada de la SMT, que indicara si el segmento esta paginado o no.

Ventajas de la segmentación paginada

El esquema de segmentación paginada tiene todas las ventajas de la segmentación y la paginación:

- Debido a que los espacios de memorias son segmentados, se garantiza la facilidad de implantar la compartición y enlace.
- Como los espacios de memoria son paginados, se simplifican las estrategias de almacenamiento.
- Se elimina el problema de la fragmentación externa y la necesidad de compactación.

Desventajas de la segmentación paginada

- Las tres componentes de la dirección y el proceso de formación de direcciones hace que se incremente el costo de su implantación. El costo es mayor que en el caso de segmentación pura o paginación pura.
- Se hace necesario mantener un número mayor de tablas en memoria, lo que implica un mayor costo de almacenamiento.

Sigue existiendo el problema de fragmentación interna de todas, o casi, todas las páginas finales de cada uno de los segmentos. Bajo paginación pura se desperdician solo la última página asignada, mientras que bajo segmentación paginada el desperdicio puede ocurrir en todos los segmentos asignados.

Memoria virtual: cuando se compararon la paginación y segmentación simple por un lado, con la partición estática y dinámica por otro, se pusieron las bases de un avance fundamental en la gestión de memoria. Las claves de este avance son dos características de la paginación y la segmentación:

- I. Todas las referencias a memoria dentro de un proceso son direcciones lógicas que se traducen dinámicamente a direcciones físicas durante la ejecución. Esto quiere decir que un proceso puede cargarse y descargarse de memoria principal de forma que ocupe regiones diferentes en instantes diferentes a lo largo de su ejecución.
- II. Un proceso puede dividirse en varias partes, páginas o segmentos, y no es necesario que estas partes se encuentren contiguas en memoria principal durante la ejecución. Esto es posible por la combinación de la traducción dinámica de direcciones en tiempo de ejecución y el uso de una tabla de páginas o de segmentos.

Si estas dos características están presentes, no será necesario que todas las páginas o todos los segmentos de un proceso estén en memoria durante la ejecución. Si tanto la parte, página o segmento, que contiene la siguiente instrucción a leer como la parte que contiene los próximos datos a acceder están en memoria principal, la ejecución podrá continua al menos por un tiempo. El sistema operativo comienza trayendo sólo unos pocos fragmentos, incluido el fragmento que contiene el comienzo del programa. Se llamará conjunto residente del proceso a la “parte” de un proceso que está realmente en memoria principal. Cuando el proceso se ejecuta, todo irá sobre ruedas mientras todas las referencias a memoria estén en posiciones que pertenezcan al conjunto residente. A través de la tabla de páginas o de segmentos, el procesador encuentra una dirección lógica que no está en memoria principal, genera una interrupción que indica que fallo de acceso a memoria. El sistema operativo pasa el proceso interrumpido al estado bloqueado y toma el control. Para que la ejecución de este se siga más tarde, el sistema operativo deberá traer a memoria principal el fragmento del proceso que contiene la dirección lógica que provocó el fallo de acceso. Para ello, el sistema operativo emite una solicitud de E/S a disco, una vez emitido la solicitud puede despachar otro proceso para que se ejecute mientras se realiza la operación de E/S. Una vez que el fragmento deseado se ha traído a memoria principal y se ha emitido la interrupción de E/S, se devuelve el control al sistema operativo, que coloca el proceso afectado en el estado de Listo.

Puede cuestionarse que en esta operación un proceso puede estar siendo ejecutado y ser interrumpido sin otra razón que un fallo en la carga de todos los fragmentos necesarios para el proceso. Por ahora, se aplazara la discusión de esta cuestión con la seguridad de que es posible alcanzar esta eficiencia. Las ventajas que presenta esta operación son dos:

- I. **Se pueden conservar más procesos en memoria principal:** puesto que se van a cargar solo algunos fragmentos de un proceso particular, habrá sitio para más procesos. Esto conduce a una utilización más eficiente del procesador, puesto que es más probable que, por lo menos, uno de los numerosos procesos este en estado Listo en un instante determinado.
- II. **Es posible que un proceso sea más grande que toda la memoria principal:** se elimina así una de las limitaciones más notorias de la programación. Sin el esquema que se ha expuesto, un programador debe ser consciente de cuanta memoria tiene disponible. Si el programa que esta escribiendo es demasiado grande, el programador debe idear formas de estructurar el programa en fragmentos que puedan cargarse de forma separada con algún tipo de estrategia de superposición. Con una memoria virtual basada en paginación

o segmentación, este trabajo queda para el sistema operativo y el hardware. En lo que atañe al programador, se las arregla con una memoria enorme, dependiendo del tamaño de almacenamiento en disco. El sistema operativo cargara automáticamente en memoria principal los fragmentos de un proceso cuando los necesita.

Para llevar a cabo la memoria virtual el hardware debe soportar paginación o segmentación por demanda, se debe disponer de un dispositivo de memoria secundaria, disco, que dé el apoyo para almacenar las secciones del proceso que no están en memoria principal, la cual se determina área de intercambio. El sistema operativo debe ser capaz de manejar el movimiento de las páginas o segmentos entre la memoria principal y la secundaria.

Memoria virtual con paginación: en paginación simple se indicó que cada proceso tiene su propia tabla de páginas y que, cuando carga todas sus páginas en memoria principal, se crea y carga en memoria principal una tabla de páginas. Cada entrada de la tabla de páginas contiene el número de marco de la página correspondiente en memoria principal. Cuando se considera un esquema de memoria virtual basado en la paginación se necesita la misma estructura, una tabla de páginas. Nuevamente, es normal asociar una única tabla de páginas con cada proceso. En este caso, las entradas de la tabla de páginas pasan a ser más complejas Puesto que solo algunas de las páginas de un proceso pueden estar en memoria principal, se necesita un bit en cada entrada de la tabla para indicar si la página correspondiente está presente, V, en memoria principal o no lo está. Si el bit indica que la pagina esta en memoria, la entrada incluye también el número de marco para esa página.

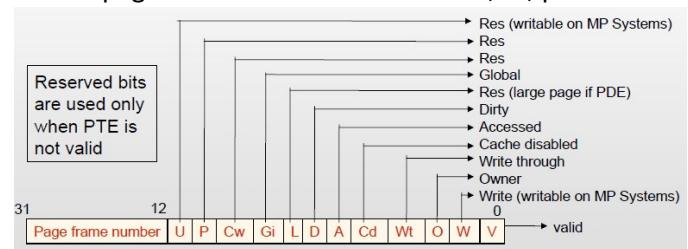
Otro bit de control necesario en la entrada de la tabla de páginas es el bit de modificación, M, para indicar si el contenido de la página

correspondiente se ha alterado desde que la página se cargó en memoria principal. Si no ha habido cambios, no es necesario escribir la página cuando sea sustituida en el marco que ocupa actualmente. Puede haber también otros bits de control. Por ejemplo, si la

protección o la compartición se gestionan a nivel de página, se necesitaran más bits con tal propósito. En la imagen se muestra una estructura de entrada en tabla de páginas de x86.

Cuando hay que cargar un proceso, el paginador realiza una estimación de qué páginas serán utilizadas antes de descargar de nuevo el proceso. En lugar de cargar el proceso completo, el paginador sólo carga en la memoria las páginas necesarias; de este modo, evita cargar en la memoria las páginas que no vayan a ser utilizadas, reduciendo así el tiempo de carga y la cantidad de memoria física necesaria.

Con este esquema, necesitamos algún tipo de soporte hardware para distinguir entre las páginas que se encuentran en memoria y las páginas que residen en el disco. Podemos usar para este propósito el esquema basado en un bit válido-inválido. Sin embargo, esta vez, cuando se configura este bit como "válido", la página asociada será legal y además se encontrará en memoria. Si el bit se configura como "inválido", querrá decir que o bien la página no es válida, es decir, no se encuentra en el espacio lógico de direcciones del proceso o es válida pero está actualmente en el disco. La entrada de la tabla de páginas correspondiente a una página que se cargue en memoria se configurará de la forma usual, pero la entrada de la tabla de páginas correspondiente a una página que no se encuentre actualmente en la memoria se marcará simplemente como inválida o contendrá la dirección de la página dentro del disco.



Marcar una página como inválida no tendrá ningún efecto si el proceso no intenta nunca acceder a dicha página. Por tanto, si nuestra estimación inicial es correcta y cargamos en la memoria todas las páginas que sean verdaderamente necesarias, y sólo esas páginas, el proceso se ejecutará exactamente igual que si hubiéramos cargado en memoria todas las páginas. Mientras que el proceso se ejecute y acceda a páginas que sean residentes en memoria, la ejecución se llevará a cabo normalmente. Pero cuando el proceso trate de acceder a una página que no haya sido cargada, el acceso a una página como inválida provoca una interrupción de **fallo de página**. El hardware de paginación, al traducir la dirección mediante la tabla de páginas, detectará que el bit de página inválida está activado, provocando una interrupción dirigida al sistema operativo.

Esta interrupción se produce como resultado de que el sistema operativo no ha cargado anteriormente en memoria la página deseada. El procedimiento para tratar este fallo de página es muy sencillo y se muestra en la figura:

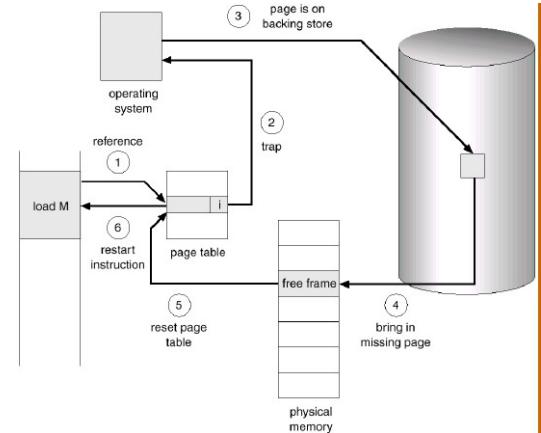
- I. Comprobamos una tabla interna, que usualmente se mantiene con el bloque del control del proceso, correspondiente a este proceso, para determinar si la referencia era un acceso de memoria válido o inválido.
- II. Si la referencia era inválida, terminamos el proceso. Si era válida pero esa página todavía no ha sido cargada, la cargamos en memoria. El sistema operativo, podrá colocar al proceso en estado de "Blocked" mientras gestiona que la página que se necesite sea cargada.
- III. Buscamos un marco libre, por ejemplo, tomando uno de la lista de marcos libres.
- IV. Ordenamos una operación de disco para leer la página deseada en el marco recién asignado. El sistema operativo puede asignarle la CPU a otro proceso mientras se completa la E/S.
- V. 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. La operación de E/S finaliza y notifica al sistema operativo; este actualiza la tabla de páginas del proceso, colocando el bit V en 1 en la página en cuestión y colocando la dirección base del frame donde se colocó la página.
- VI. Reiniciamos la instrucción que fue interrumpida, el proceso pasa a estado de Ready. El proceso podrá ahora acceder a la página como si siempre hubiera estado en memoria.

La paginación bajo demanda puede afectar significativamente al rendimiento de un sistema informático. Vamos a calcular el tiempo de acceso efectivo a una memoria, EAT, con paginación bajo demanda. Para la mayoría de los sistemas informáticos, el tiempo de acceso a memoria, que designaremos ma , va de 10 a 200 ns. Mientras no tengamos fallos de página, el tiempo de acceso efectivo será igual al tiempo de acceso a memoria. Sin embargo, si se produce un fallo de página, deberemos primero leer la página relevante desde el disco y luego acceder a la página deseada. Sea p la probabilidad de que se produzca un fallo de página, $0 < p < 1$. Cabe esperar que p esté próximo a cero, es decir, que sólo vayan a producirse unos cuantos fallos de página. El tiempo de acceso efectivo será entonces:

$$EAT = (1 - p) \cdot ma + p \cdot \text{tiempo de fallo de página}$$

Para calcular el tiempo de acceso efectivo, debemos conocer cuánto tiempo se requiere para dar servicio a un fallo de página. Cada fallo de página hace que se produzca la siguiente secuencia:

- I. Interrupción al sistema operativo.



- II. Guardar los registros de usuario y el estado del proceso.
- III. Determinar que la interrupción se debe a un fallo de página.
- IV. Comprobar que la referencia de página era legal y determinar la ubicación de la página en el disco.
- V. Ordenar una lectura desde el disco para cargar la página en un marco libre:
 - Esperar en la cola correspondiente a este dispositivo hasta que se dé servicio a la solicitud de lectura.
 - Esperar el tiempo de búsqueda o latencia del dispositivo.
 - Comenzar la transferencia de la página hacia un marco libre.
- VI. Mientras estamos esperando, asignar la CPU a algún otro usuario.
- VII. Recibir una interrupción del subsistema de E/S de disco.
- VIII. Guardar los registros y el estado del proceso para el otro usuario, si se ejecuta 6.
- IX. Determinar que la interrupción corresponde al disco.
- X. Corregir la tabla de páginas y otras tablas para mostrar que la página deseada se encuentra ahora en memoria.
- XI. Esperar a que se vuelva a asignar la CPU a este proceso.
- XII. Restaurar los registros de usuario, el estado del proceso y la nueva tabla de páginas y reanudar la ejecución de la instrucción interrumpida.

No todos los pasos son necesarios, en cualquier caso siempre nos enfrentamos a tres componentes principales:

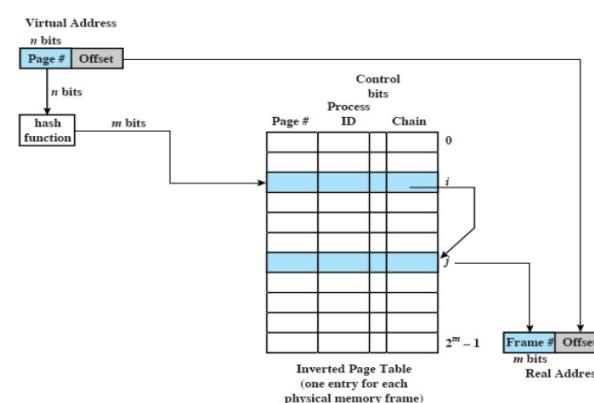
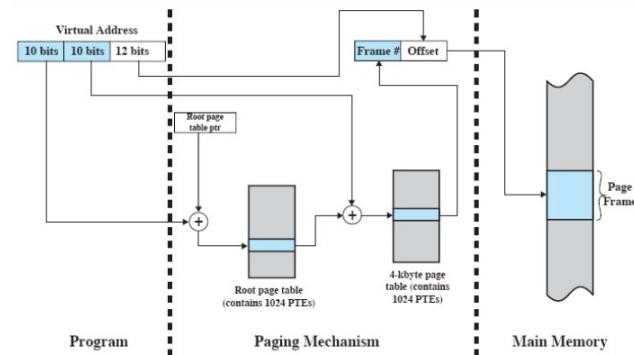
- I. Servir la interrupción de fallo de página.
- II. Leer la página.
- III. Reiniciar el proceso.

Nota: ejemplo de EAT en diapositiva Memoria 3 diapositiva N°17.

Estructuras de la tabla de páginas: el mecanismo básico de lectura de una palabra de memoria supone la traducción por medio de la tabla de páginas de una dirección virtual o lógica, formada por un número de página y un desplazamiento, a una dirección física que está formada por un número de marco y un desplazamiento. Puesto que la tabla de páginas es de longitud variable, en función del tamaño del proceso, no es posible suponer que quepa en los registros. En su lugar, debe estar en memoria principal para ser accesible. La figura sugiere una implementación en hardware de este esquema. Cuando se está ejecutando un proceso en particular, la dirección de comienzo de la tabla de páginas para este proceso se mantiene en un registro. El número de página de la dirección virtual se emplea como índice en esta tabla para buscar el número de marco correspondiente. Este se combina con la parte de desplazamiento de la dirección virtual para generar la dirección real deseada.

En la mayoría de los sistemas hay una tabla de páginas por proceso, pero cada proceso puede ocupar una cantidad enorme de memoria virtual.

Evidentemente, la cantidad de memoria dedicada solo a la tabla de páginas puede ser inaceptable. Para solucionar este problema, la mayoría de los esquemas de memoria virtual almacenan las tablas de páginas en memoria virtual en vez de en memoria



real. Esto significa que estas tablas de páginas están también sujetas a paginación, de la misma forma que las otras páginas. Cuando un proceso está ejecutando, al menos una parte de su tabla de páginas debe estar en memoria principal, incluyendo la entrada de la tabla de páginas para la página actualmente en ejecución. Algunos procesadores usan un esquema a dos niveles para organizar grandes tablas de páginas. En este esquema, hay un directorio de páginas en el que cada entrada señala a una tabla de páginas. Así pues, si la longitud del directorio de páginas es X y la longitud máxima de una tabla de páginas es Y , un proceso puede estar formado por hasta $X \times Y$ páginas. Normalmente, la longitud máxima de una tabla de páginas está limitada a una página. Un enfoque alternativo al uso de tablas de páginas a uno o dos niveles consiste en el uso de una estructura de **tabla de páginas invertida**, como se ve en la figura. Con este método, la parte del número de página en una dirección virtual se contrasta en una tabla de dispersión por medio de una función de dispersión simple. La tabla de dispersión contiene un puntero a la tabla de páginas invertida, que contiene a su vez las entradas de la tabla de páginas. Con esta estructura, hay una entrada en la tabla de dispersión y la tabla de páginas invertida por cada página de memoria real en lugar de una por cada página virtual. Así pues, se necesita una parte fija de la memoria real para las tablas, sin reparar en el número de procesos o de páginas virtuales soportados. Puesto que más de una dirección de memoria pueden corresponderse con la misma entrada de la tabla de dispersión, para gestionar las colisiones se emplea una técnica de encadenamiento. La técnica de dispersión genera normalmente cadenas cortas, de dos a tres entradas cada una.

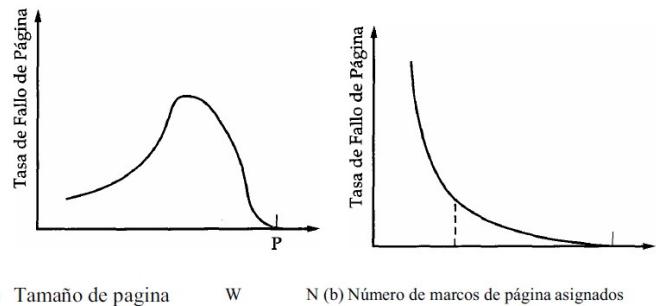
En resumen las formas de organización son:

- Tabla de nivel 1, tabla única lineal.
- Tabla de 2 niveles o más, tabla multinivel.
- Tabla invertida, hashing.

La forma de organizarla depende del hardware subyacente.

Tamaño de página: una decisión importante de diseño del hardware es el tamaño de página que se va a usar. Hay varios factores que considerar. Uno es la fragmentación interna. Sin duda, cuanto menor sea el tamaño de página, menor será la cantidad de fragmentación interna. Para optimizar el uso de la memoria principal, es positivo reducir la fragmentación interna. Por otro lado, cuanto menor sea la página, mayor será el número de páginas que se necesitan por proceso. Un número mayor de páginas por proceso significa que las tablas de páginas serán mayores. Para programas grandes, en un entorno multiprogramado intensivo, esto puede significar que una gran parte de las tablas de páginas de los procesos activos deban estar en memoria virtual, no en memoria principal. Así pues, pueden suceder dos fallos de página para una única referencia a memoria: primero, para traer la parte necesaria de la tabla de páginas y, segundo, para traer la página del proceso. Otro factor radica en que las características físicas de la mayoría de los dispositivos de memoria secundaria, que son de rotación, son propicias a tamaños de página mayores, puesto que así se obtiene una transferencia por bloques de datos que es más eficiente.

Para complicar la cuestión se tiene el efecto que tiene el tamaño de página en el porcentaje de fallos de página. Este comportamiento se representa, en líneas generales, en la figura y se basa en el principio de cercanía. Si el tamaño de página es muy pequeño, normalmente estarán disponibles en memoria principal un gran número de páginas para cada proceso. Después de un tiempo, todas las páginas en memoria contendrán parte de las referencias más recientes del



(a) Tamaño de pagina

W

N

(b)

Número de marcos de página asignados

proceso. Así pues, la tasa de fallos de página será menor. Cuando se incrementa el tamaño de la página, cada página individual contendrá posiciones cada vez más distantes de cualquier referencia reciente. Así pues, se atenúa el efecto del principio de cercanía y comienza a aumentar la tasa de fallos de página. Sin embargo, la tasa de fallos de página comenzara a bajar cuando, finalmente, el tamaño de página se aproxime al tamaño de todo el proceso, punto P del diagrama. Cuando una única página abarca todo el proceso, no hay fallos de página.

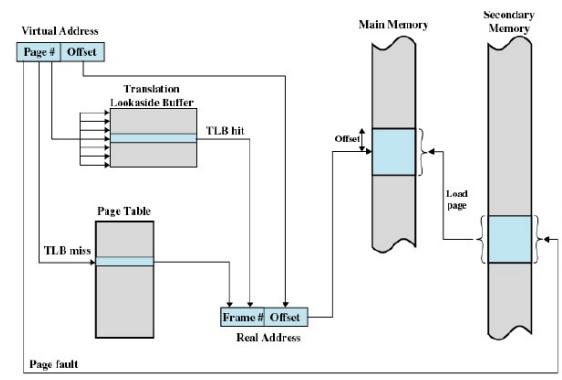
Una dificultad mas es que la tasa de fallos de página viene determinada también por el número de marcos asignados a un proceso. La figura b demuestra que, para un tamaño de página fijo, la tasa de fallos baja conforme sube el número de páginas contenidas en memoria principal. Así pues, una política del software, la cantidad de memoria asignada a cada Proceso, afectara a una decisión de diseño del hardware.

Por último, el diseño del tamaño de página está relacionado con el tamaño de la memoria física principal. Las implementaciones de memoria virtual en la mayoría de los sistemas actuales fueron diseñadas bajo el supuesto de que la memoria física no es muy grande, normalmente comprendida en un rango de 4M a 256M bytes. Sin embargo, en los próximos años se pueden esperar que aparezcan tamaños de memoria física mayores en estaciones de trabajo y grandes maquinas.

Translation lookaside buffer, TLB: en principio, cada referencia a memoria virtual puede generar dos accesos a memoria: uno para obtener la entrada de la tabla de páginas correspondiente y otro para obtener el dato deseado. Así pues, un esquema sencillo de memoria virtual podría tener el efecto de doblar el tiempo de acceso a memoria. Para solucionar este problema, la mayoría de los esquemas de memoria virtual hacen uso de una cache especial para las entradas de la tabla de páginas, llamada generalmente **buffer de traducción adelantada, TLB**. Esta cache funciona del mismo modo

que una memoria cache y contiene aquellas entradas de la tabla de páginas usadas hace menos tiempo. La organización del hardware de paginación resultante se muestra en la figura, dada una dirección virtual, el procesador examinara primero la TLB. Si la entrada de tabla de páginas buscada está presente, un acierto en la TLB, se obtiene el número de marco y se forma la dirección real. Si no se encuentra la entrada de la tabla de páginas buscada, un fallo de TLB, el procesador emplea el número de página para buscar en la tabla de páginas del proceso y examinar la entrada correspondiente de la tabla de páginas. Si se encuentra activo el bit de presencia, es que la página esta en memoria principal y el procesador puede obtener el número de marco de la entrada de la tabla de páginas para formar la dirección real. El procesador, además, actualiza la TLB para incluir esta nueva entrada de la tabla de páginas. Por último, si el bit de presencia no está activo, es que la página buscada no está en memoria principal y se produce un fallo en el acceso a memoria, llamado **fallo de página**. En este punto, se abandona el ámbito del hardware y se invoca al sistema operativo, que carga la página necesaria y actualiza la tabla de páginas.

La figura muestra un diagrama de flujo que representa el uso de la TLB. El diagrama muestra que si una página no está en memoria principal, una interrupción de fallo de página hace que se llame a la rutina de gestión de fallos de página. Para mantener la sencillez del diagrama, no se tiene en cuenta el hecho de que el sistema operativo puede pasar a ejecutar otro proceso mientras se realiza la operación de E/S a disco. Debido al principio de cercanía, la mayoría de las referencias a



memoria virtual se situaran en las páginas usadas recientemente. Por tanto, la mayoría de las referencias van a involucrar a las entradas de la tabla de páginas en la cache.

Existe una serie de detalles adicionales sobre la organización real de la TLB.

Puesto que la TLB contiene solo algunas de las entradas de la tabla de páginas completa, no se puede indexar simplemente la TLB por el número de página. En su lugar, cada entrada de la TLB debe incluir el número de página, además de la entrada completa a la tabla de páginas. El procesador estará equipado con hardware que permita consultar simultáneamente varias entradas de la TLB para determinar si hay una coincidencia en el número de página. Esta

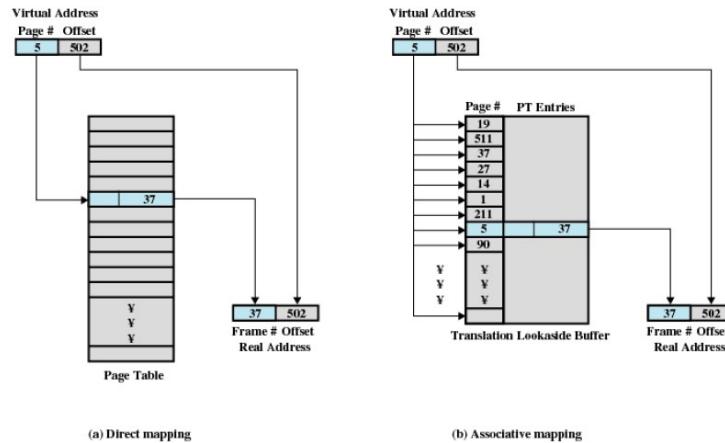
técnica llamada *correspondencia asociativa* y contrasta con la correspondencia directa, que se emplea para buscar en la tabla de páginas. El diseñador de la TLB debe tener también en cuenta la forma en que se organizan las entradas en la TLB y que entrada reemplazar cuando se introduce una nueva. Estas cuestiones deben considerarse en cualquier diseño de cache de hardware.

Por último, el mecanismo de memoria virtual debe interactuar con el sistema de cache, no con la cache TLB, sino con la cache de memoria principal. Una dirección virtual tendrá normalmente un formato de número de página más desplazamiento. En primer lugar, el sistema de memoria consulta la TLB para ver si encuentra la entrada de la tabla de páginas correspondiente. Si es así, la dirección real física se genera combinando el número de marco con el desplazamiento. Si no, se accede a la entrada de la tabla de páginas. Una vez que se tiene la dirección real, que está en forma de una etiqueta y un resto, se consulta la cache para ver si está presente el bloque que contiene dicha palabra. Si lo está, es devuelto a la CPU. Si no, se toma la palabra de memoria principal.

Se puede apreciar la complejidad del hardware de la CPU involucrado en una única referencia a memoria. Traducir la dirección virtual a una dirección real supone hacer referencia a una entrada de la tabla de páginas, que puede estar en la TLB, en memoria principal o en disco. La palabra referenciada puede estar en cache, en memoria principal o en disco. En este último caso, debe cargarse en memoria principal la página que contiene la palabra y su bloque en la cache. Además, se debe actualizar la entrada de la tabla de páginas para dicha página.

Asignación de marcos: con memoria virtual paginada no resulta necesario y, de hecho, puede no ser posible, traer todas las páginas de un proceso a memoria principal para preparar su ejecución. Así pues, el sistema operativo debe decidir cuantas páginas traer, es decir, cuanta memoria principal asignar a un determinado proceso. Aquí entran en juego varios factores:

- Cuanto menor es la cantidad de memoria asignada a un proceso, mayor es el número de procesos que pueden estar en memoria principal en cualquier instante. Esto aumenta la probabilidad de que el sistema operativo encuentre al menos un proceso Listo en cualquier instante dado y, por tanto, reduzca el tiempo perdido en el intercambio.
- Si en memoria principal hay un número relativamente pequeño de páginas de un proceso, entonces, a pesar del principio de cercanía, el porcentaje de fallos de página será algo mayor.



- Por encima de un determinado tamaño, la asignación de memoria adicional a un proceso en particular no tendrá efectos notables en el porcentaje de fallos de página para ese proceso, debido al principio de cercanía.

Con estos factores en mente, en los sistemas operativos actuales se pueden encontrar dos tipos de políticas:

- I. **Asignación fija:** otorga a cada proceso un número fijo de páginas en las que ejecutar. Dicho número se decide en el instante de carga inicial, instante de creación del proceso, y puede estar determinado por el tipo de proceso, interactivo, por lotes, tipo de aplicación, o en función de las directrices del programador o del administrador del sistema. Cada vez que se produce un fallo de página en la ejecución de un proceso, se debe reemplazar una de las páginas de dicho proceso por la página que se necesite.
- I. **Asignación dinámica:** permite que el número de marcos asignados a un proceso cambie a lo largo de su vida. En el mejor de los casos, un proceso que está sufriendo de forma permanente un alto número de fallos de página, demostrando que el principio de cercanía apenas se cumple para él, recibirá marcos adicionales para reducir el porcentaje de fallos de página, mientras que un proceso con una tasa de fallos excepcionalmente baja, que demuestra que el proceso se comporta bastante bien desde el punto de vista de la cercanía, verá reducida su asignación, con la esperanza de que esto no aumentara demasiado su tasa de fallos. El uso de una política de asignación variable está relacionado con el concepto de alcance del reemplazo. Está parece ser más potente, sin embargo, la dificultad de este método está en que requiere que el sistema operativo evalúe el comportamiento de los procesos activos. Esto produce inevitablemente la necesidad de más software en el sistema operativo y es dependiente de los mecanismos de hardware ofrecidos por la plataforma del procesador.

Alcance del reemplazo: puede clasificarse en global o local. Ambos tipos de políticas son activadas por un fallo de página que se produce cuando no hay marcos libres. Para seleccionar la página a reemplazar, una **política de reemplazo local** escoge únicamente de entre las páginas residentes del proceso que origino el fallo de página. Una **política de reemplazo global** considera todas las páginas en memoria como candidatas para reemplazar, independientemente del proceso al que pertenezcan. Aunque las políticas locales son más fáciles de analizar, no hay ninguna evidencia de que se comporten mejor que las políticas globales, las cuales son atrayentes por su simplicidad de implementación y su mínimo coste:

- **Asignación fija y alcance local:** se tiene un proceso que ejecuta en memoria principal con un número fijo de páginas. Cuando se produce un fallo de página, el sistema operativo debe elegir la página a reemplazar entre las de dicho proceso que están actualmente en memoria. Con esta política es necesario decidir por anticipado la cantidad de memoria asignada a un proceso. Esta decisión puede hacerse en función del tipo de aplicación y de la cantidad solicitada por el programa. Las desventajas de esta solución son dos: Si la asignación tiende a ser demasiado pequeña, se producirá un alto porcentaje de fallos de página, haciendo que el sistema multiprogramado al completo funcione lentamente. Si la asignación tiende a ser innecesariamente grande, habrá muy pocos programas en memoria principal y el procesador estará desocupado un tiempo considerable o bien se consumirá un tiempo importante en intercambio.
- **Asignación variable y alcance global:** esta combinación es la más sencilla de implementar y ha sido adoptada por un buen número de sistemas operativos. En un instante dado, en memoria principal habrá varios procesos, cada uno de ellos con un cierto número de marcos asignados. Normalmente, el sistema operativo también mantiene una lista de

marcos libres. Cuando se produce un fallo de página, se añade un marco libre al conjunto residente del proceso y se carga la página. Así pues, los procesos que producen fallos de página incrementan gradualmente su tamaño, lo que ayuda a reducir el número global de fallos de página en el sistema.

La dificultad de este método está en la elección del reemplazo. Cuando no hay marcos libres, el sistema operativo debe elegir una página que este en memoria para reemplazar. La selección se realiza entre todas las páginas de memoria, excepto los marcos bloqueados, como los del núcleo. Usando cualquiera de las políticas tratadas en el apartado anterior, la página elegida puede pertenecer a cualquier proceso residente; no hay ninguna disciplina que determine el proceso que debe perder una página de su conjunto residente. Es más, la selección del proceso que sufre la reducción en el conjunto residente puede que no sea óptima.

Una forma de contrarrestar los problemas potenciales de rendimiento de una política de asignación variable y alcance global es el almacenamiento intermedio de páginas. De esta manera, la elección de la página a reemplazar se hace menos significativa, ya que la página puede ser recuperada si se produce una referencia antes de que se escriba el bloque con el siguiente grupo de páginas.

Algoritmos de reemplazo: además de la estrategia de gestión del conjunto residente, existen ciertos algoritmos básicos que se emplean para la selección de una página a reemplazar:

- **Optimo:** selecciona para reemplazar la página que tiene que esperar una mayor cantidad de tiempo hasta que se produzca la referencia siguiente. Se puede demostrar que esta política genera el menor número de fallos de página. Sin duda, este algoritmo resulta imposible de implementar, puesto que requiere que el sistema operativo tenga un conocimiento exacto de los sucesos futuros. Sin embargo, sirve como un estándar con el que comparar los otros algoritmos.
- **FIFO, first in first out:** el sistema operativo sólo tiene que guardar en orden las páginas que fueron cargadas, de modo que al necesitar hacer espacio pueda fácilmente elegir la primera página cargada. Se usa una cola, al cargar una página nueva se ingresa en el último lugar. Aunque las colas FIFO son simples e intuitivas, no se comportan de manera aceptable en la aplicación práctica, por lo que es raro su uso en su forma simple. Uno de los problemas que presentan es la llamada Anomalía FIFO o Anomalía de Belady. Belady encontró ejemplos en los que un sistema con un número de marcos de páginas igual a tres tenía menos fallos de páginas que un sistema con cuatro marcos de páginas. El problema consiste en que podemos quitar de memoria una página de memoria muy usada, sólo porque es la más antigua.
- **LRU, least recently used:** reemplaza la página de memoria que no ha sido referenciada desde hace más tiempo. Debido al principio de cercanía, esta debería ser la página con menor probabilidad de ser referenciada en un futuro cercano. De hecho, la política LRU afina casi tanto como la política óptima. El problema de este método es su dificultad de implementación. Una solución sería etiquetar cada página con el instante de su última referencia; esto tendría que hacerse para cada referencia a memoria, tanto para instrucciones como datos. Incluso si el hardware respaldara este esquema, la sobrecarga resultaría tremenda. Como alternativa, se podría mantener una pila de referencias a página, aunque también con un coste elevado.
- **Second Chance:** es un algoritmo FIFO pero evita deshacerse de una página de uso frecuente, hace uso del bit R inspeccionándolo de tal forma que, si es 0, la página es antigua y no utilizada, por lo que, en caso de fallo de página, es reemplazada de manera

inmediata. En caso de que dicho bit sea 1, es cambiado a cero y se cambia al final de la lista de páginas como si hubiera llegado en ese momento a la memoria. Luego continúa la búsqueda siguiendo lo que avanza en la lista.

- **NRU, not recently used:** es uno de los esquemas que intentan aproximarse al algoritmo LRU. Específicamente es una modificación del algoritmo de segunda oportunidad, el cual considera el bit de referencia R y el bit de modificación M como un par ordenado (R, M) respectivamente.

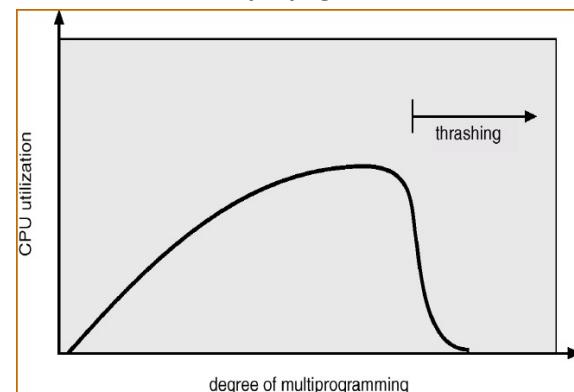
Thrashing: en memoria solo se tienen unos pocos fragmentos de un proceso dado y, por tanto, se pueden mantener más procesos en memoria. Es más, se ahorra tiempo, porque los fragmentos que no se usan no se cargan ni se descargan de memoria. Sin embargo, el sistema operativo debe saber cómo gestionar este esquema. En un estado estable, prácticamente toda la memoria principal estará ocupada con fragmentos de procesos, por lo que el procesador y el sistema operativo tendrán acceso directo a la mayor cantidad de procesos posible. Así pues, cuando el sistema operativo traiga a memoria un fragmento, deberá expulsar otro. Si expulsa un fragmento justo antes de ser usado, tendrá que traer de nuevo el fragmento de manera casi inmediata.

Demasiados intercambios de fragmentos conducen a lo que se conoce como **hiperpaginación, thrashing**. El procesador consume más tiempo intercambiando fragmentos que ejecutando instrucciones de usuario. Como consecuencia, hay una naja importante de performance en el sistema. Algunos algoritmos hacen que el sistema operativo, intente adivinar, en función de la historia reciente, qué fragmentos se usarán con menor probabilidad en un futuro próximo.

El fenómeno de la hiperpaginación se ilustra en la figura, cuando el grado de multiprogramación supera un pequeño valor, se podría esperar que la utilización del procesador aumentara, puesto que hay menos posibilidades de que todos los procesos estén bloqueados. Sin embargo, se alcanza un punto en el que el conjunto residente en promedio no es adecuado. En este punto, el número de fallos de página se eleva drásticamente y la utilización del procesador se desploma.

Hay varias formas de abordar este problema. Los algoritmos del conjunto de trabajo o de frecuencia de fallos de página incorporan implícitamente el control de carga. Solo pueden ejecutar aquellos procesos cuyos conjuntos residentes sean suficientemente grandes. Dado un tamaño exigido para el conjunto residente de cada proceso activo, la política determina automáticamente y dinámicamente el número de programas activos.

El sistema operativo monitoriza la utilización de la CPU. Si esa tasa de utilización es demasiado baja, se incrementa el grado de multiprogramación introduciendo un nuevo proceso en el sistema. Se utiliza un algoritmo de sustitución global de páginas, que sustituye las páginas sin tomar en consideración a qué proceso pertenecen. Ahora suponga que un proceso entra en una nueva fase de ejecución y necesita más marcos de memoria. El proceso comenzará a generar fallos de página y a quitar marcos a otros procesos. Dichos procesos necesitan, sin embargo, esas páginas por lo que también generan fallos de página, quitando marcos a otros procesos. Los procesos que generan los fallos de páginas deben utilizar los procesos de paginación para cargar y descargar las páginas en memoria y, a medida que se ponen en cola para ser servidos por el dispositivo de paginación, la cola de procesos preparados se vacía. Como los procesos están a la espera en el dispositivo de paginación, la tasa de utilización de la CPU disminuye.



El planificador de la CPU ve que la tasa de utilización de la CPU ha descendido e *incrementa* como resultado el grado de multiprogramación. El nuevo proceso tratará de iniciarse quitando marcos "a los procesos que se estuvieran ejecutando, haciendo que se provoquen más fallos de página y que la cola del dispositivo de paginación crezca. Como resultado, la utilización de la CPU cae todavía más y el planificador de la CPU trata de incrementar el grado de multiprogramación todavía en mayor medida. Se ha producido una hiperpaginación y la tasa de procesamiento del sistema desciende vertiginosamente, a la vez que se incrementa enormemente la tasa de fallos de página. Como resultado, también se incrementa el tiempo efectivo de acceso a memoria y no se llegará a realizar ningún trabajo útil, porque los procesos invertirán todo su tiempo en los mecanismos de paginación.

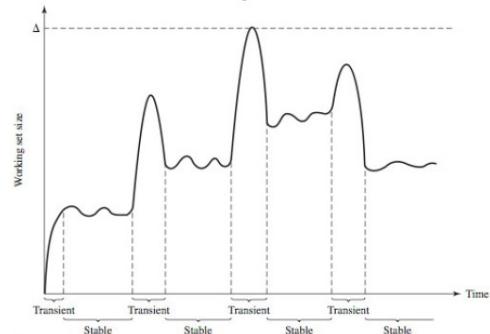
A medida que se incrementa el grado de multiprogramación, también lo hace la tasa de utilización de la CPU, aunque más lentamente, hasta alcanzar su máximo. Si el grado de multiprogramación se incrementa todavía más, aparece la hiperpaginación y la tasa de utilización de la CPU cae abruptamente. En este punto, para incrementar la tasa de utilización de la CPU y poner fin a la hiperpaginación, es necesario *reducir* el grado de multiprogramación.

Podemos limitar los efectos de la hiperpaginación utilizando un algoritmo de sustitución local, o un algoritmo de sustitución basado en prioridades. Con la sustitución local, si uno de los procesos entra en hiperpaginación, no puede robar marcos de otro proceso y hacer que éste también entre en hiperpaginación. Sin embargo, esto no resuelve el problema completamente. Si los procesos están en hiperpaginación, se pasarán la mayor parte del tiempo en la cola del dispositivo de paginación. De este modo, el tiempo medio de servicio para un fallo de página se incrementará debido a que ahora el tiempo medio invertido en la cola del dispositivo de paginación es mayor. Por tanto, el tiempo de acceso efectivo se incrementará incluso para los procesos que no estén en hiperpaginación.

Una forma de prevenir la hiperpaginación es por la técnica del modelo de localidad.

Modelo de localidad: en la figura se muestra cómo puede variar el tamaño del conjunto de trabajo a lo largo del tiempo para un valor fijo de A. El modelo de localidad afirma que, a medida que un proceso se ejecuta, se va desplazando de una localidad a otra. Una localidad es un conjunto de páginas que se utilizan activamente de forma combinada. Todo programa está generalmente compuesto de varias localidades diferentes, que pueden estar solapadas. Por ejemplo, cuando se invoca una función, ésta define una nueva localidad. En esta localidad, las referencias a memoria se hacen a las instrucciones de la llamada a función, a sus variables locales y a un subconjunto de las variables globales. Cuando salimos de la función, el proceso abandona esta localidad, ya que las variables locales y las instrucciones de la función dejan de ser activamente utilizadas, aunque podemos volver a esta localidad posteriormente.

Por tanto, vemos que las localidades están definidas por la estructura del programa y por sus estructuras de datos. El modelo de localidad afirma que todos los programas exhibirán esta estructura básica de referencias a memoria. Si asignamos a un proceso los suficientes marcos como para acomodar su localidad actual. El proceso generará fallos de página para todas las páginas de su localidad, hasta que todas ellas estén en memoria; a partir de ahí, no volverá a generar fallos de página hasta que cambie de localidad. Si asignamos menos marcos que el



tamaño de la localidad actual, el proceso entrará en hiperpaginación, ya que no podrá mantener en memoria todas las páginas que esté utilizando activamente.

Working set: el modelo del conjunto de trabajo está basado en la suposición de la localidad de ejecución de los programas. Este modelo utiliza un parámetro, Δ , para definir la ventana del conjunto de trabajo. La idea consiste en examinar las Δ referencias de página más recientes. El conjunto de páginas en las Δ referencias de páginas más recientes es el conjunto de trabajo. Si una página está siendo usada de forma activa, se encontrará dentro del conjunto de trabajo. Si ya no está siendo utilizada, será eliminada del conjunto de trabajo Δ unidades de tiempo después de la última referencia que se hiciera a la misma. Por tanto, el conjunto de trabajo es una aproximación de la localidad del programa.

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

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

$$D = \sum WSS_i$$

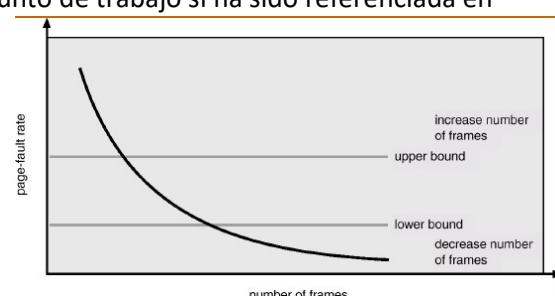
Donde D es la demanda total de marcos. Cada proceso estará utilizando activamente las páginas de su conjunto de trabajo. Así, el proceso i necesita WSS_i marcos. Si la demanda total es superior al número total de marcos disponibles, $D > m$, se producirá una hiperpaginación, porque algunos procesos no dispondrán de los suficientes marcos.

Una vez seleccionada Δ , la utilización del modelo del conjunto de trabajo es muy simple. El sistema operativo monitoriza el conjunto de trabajo de cada proceso y asigna a ese conjunto de trabajo los suficientes marcos como para satisfacer los requisitos de tamaño del conjunto de trabajo. Si hay suficientes marcos adicionales, puede iniciarse otro proceso. Si la suma de los tamaños de los conjuntos de trabajo se incrementa, hasta exceder el número total de marcos disponibles, el sistema operativo seleccionará un proceso para suspenderlo. Las páginas de ese proceso se escribirán, descargarán, y sus marcos se reasignarán a otros procesos. El proceso suspendido puede ser reiniciado posteriormente.

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

La dificultad inherente al modelo del conjunto de trabajo es la de controlar cuál es el conjunto de trabajo de cada proceso. La ventana del conjunto de trabajo es una ventana móvil. Con cada referencia a memoria, aparece una nueva referencia en un extremo y la referencia más antigua se pierde por el otro. Una página se encontrará en el conjunto de trabajo si ha sido referenciada en cualquier punto dentro de la ventana del conjunto de trabajo.

El modelo de conjunto de trabajo resulta muy adecuado y el conocimiento de ese conjunto de trabajo puede resultar útil para la paginación, pero parece una forma un tanto torpe de controlar la hiperpaginación. Hay una estrategia más directa que



está basada en la frecuencia de fallos de página, PFF, page-fault frequency.

El problema específico es cómo prevenir la hiperpaginación. Cuando se entra en hiperpaginación se produce una alta tasa de fallos de página; por tanto, lo que queremos es controlar esa alta tasa. Cuando es demasiado alta, sabemos que el proceso necesita más marcos; a la inversa, si la tasa de fallos de página es demasiado baja, puede que el proceso tenga demasiados marcos asignados. Podemos establecer sendos límites superior e inferior de la tasa deseada de fallos de página. Si la tasa real de fallos de página excede del límite superior, asignamos al proceso otro marco, mientras que si esa tasa cae por debajo del límite inferior, eliminamos un marco del proceso. Por tanto, podemos medir y controlar directamente la tasa de fallos de página para evitar la hiperpaginación. Al igual que con la estrategia del conjunto de trabajo, puede que tengamos, que suspender algún proceso. Si la tasa de fallos de página se incrementa y no hay ningún marco libre disponible, deberemos seleccionar algún proceso y suspenderlo. Los marcos liberados se distribuirán entonces entre los procesos que tengan una mayor tasa de fallos de página.

Demonio de paginación: la paginación funciona mejor cuando hay muchos marcos de página libres que se pueden reclamar al momento en que ocurran fallos de página. Si cada marco de página está lleno y además modificado, antes de que se pueda traer una nueva página se debe escribir una página anterior en el disco.

Para asegurar una provisión abundante de marcos de página libres, muchos sistemas de paginación tienen un proceso en segundo plano conocido como **demonio de paginación**, que está inactivo la mayor parte del tiempo pero se despierta en forma periódica para inspeccionar el estado de la memoria. Si hay muy pocos marcos de página libres, el demonio de paginación empieza a seleccionar páginas para desalojarlas mediante cierto algoritmo de reemplazo de páginas. Si estas páginas han sido modificadas después de haberse cargado, se escriben en el disco.

En cualquier caso se recuerda el contenido anterior de la página. En caso de que una de las páginas desalojadas se necesite otra vez antes de que se sobrescriba su marco, puede reclamarse si se elimina de la reserva de marcos de página libres. Al mantener una provisión de marcos de página a la mano se obtiene un mejor rendimiento que al utilizar toda la memoria y después tratar de encontrar un marco al momento de necesitarlo. Cuando menos el demonio de paginación asegura que todos los marcos libres estén limpios, por lo que no se necesitan escribir en el disco en un apuro a la hora de ser requeridos.

Una manera de implementar esta política de limpieza es mediante un reloj con dos manecillas. La manecilla principal es controlada por el demonio de paginación. Cuando apunta a una página sucia, esa página se escribe de vuelta al disco y la manecilla principal se avanza. Cuando apunta a una página limpia, sólo se avanza. La manecilla secundaria se utiliza para reemplazar páginas, como en el algoritmo de reloj estándar. Sólo que ahora, la probabilidad de que la manecilla secundaria llegue a una página limpia se incremente debido al trabajo del demonio de paginación. Los procesos demonios son creados por el sistema operativo durante el arranque, que apoya a la administración de la memoria. Se ejecutan cuando el sistema tiene una baja utilización o algún parámetro de la memoria lo indica, como por ejemplo, poca memoria libre o mucha memoria modificada. Las tareas que suelen llevar a cabo son:

- Limpiar las páginas modificadas sincronizándolas con el swap.
- Reducen el tiempo de swap posteriormente ya que las páginas están limpias.
- Pueden sincronizar varias páginas contiguas reduciendo el tiempo total de transferencia.
- Mantienen el número de páginas libres en el sistema a un cierto número.
- No liberan memoria del todo hasta que haga falta realmente.

Dos ejemplos de procesos demonios son el **kswapd**, proceso demonio de Linux, y el **system**, proceso demonio de Windows.

Memoria compartida: la idea es permitir que varios procesos comparten a través de la red un conjunto de páginas, posiblemente, pero no es necesario, como un solo espacio de direcciones lineal compartido. Cuando un proceso hace referencia a una página que no está asociada, obtiene un fallo de página. El manejador de fallos de página, que puede estar en espacio de kernel o de usuario, localiza entonces la máquina que contiene la página y le envía un mensaje pidiéndole que la desasocie y la envíe a través de la red. Cuando llega la página, se asocia y la instrucción que falló se reinicia.

Una ventaja de la paginación es la posibilidad de compartir código común. Esta consideración es particularmente importante en un entorno de tiempo compartido. Considere un sistema que de soporte a 40 usuarios, cada uno de los cuales esté ejecutando un editor de texto. Si el editor de texto está compuesto por 150 KB de código y 50 KB de espacio de datos, necesitaremos 8000 KB para dar soporte a los 40 usuarios. Sin embargo, si se trata de código reentrant o código puro, podrá ser compartido, como se muestra en la figura. En ella podemos ver un editor de tres páginas, cada página tiene 50 KB de tamaño, utilizándose un tamaño de página tan grande para simplificar la figura, compartido entre los tres procesos.

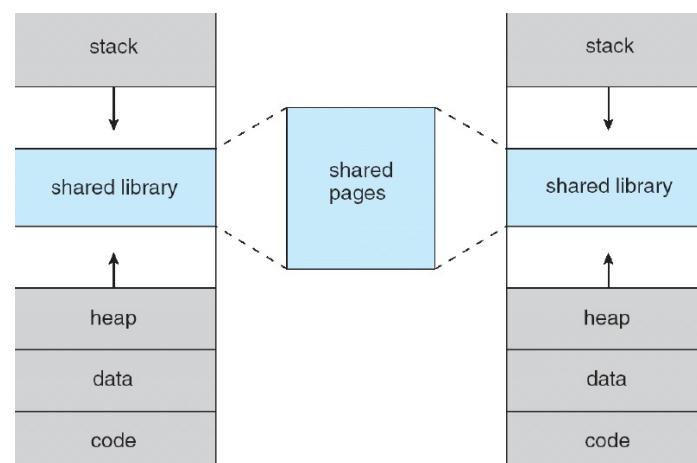
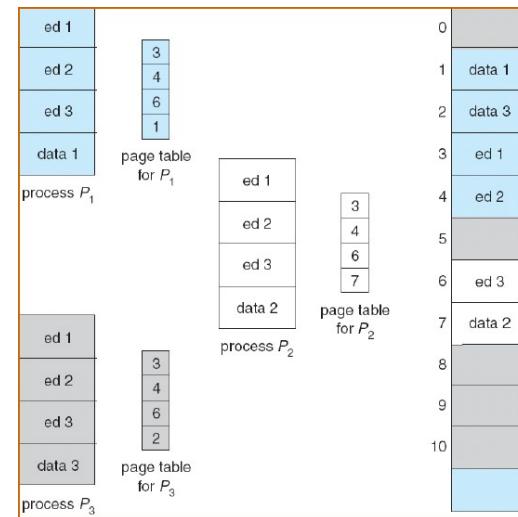
Cada proceso tiene su propia página de datos.

El código reentrant es código que no se auto-modifica; nunca cambia durante la ejecución. De esta forma, dos o más procesos pueden ejecutar el mismo código al mismo tiempo. Cada proceso tendrá su propia copia de los registros y del almacenamiento de datos, para albergar los datos requeridos para la ejecución del proceso. Por supuesto, los datos correspondientes a dos procesos distintos serán diferentes.

Sólo es necesario mantener en la memoria física una copia del editor. La tabla de páginas de cada usuario se corresponderá con la misma copia física del editor, pero las páginas de datos se harán corresponder con marcos diferentes. Así, para dar soporte a 40 usuarios, sólo necesitaremos una copia del editor 150 KB, más 40 copias del espacio de datos de 50 KB de cada usuario. El espacio total requerido será ahora de 2.150 KB, en lugar de 8.000 KB, lo que representa un ahorro considerable.

También pueden compartirse otros programas que se utilicen a menudo, como por ejemplo compiladores, sistemas de ventanas, bibliotecas de tiempo de ejecución, sistemas de base de datos, etc. Para poder compartirlo, el código debe ser reentrant. El carácter de sólo-lectura del código compartido no debe dejarse a expensas de la corrección de ese código, sino que el propio sistema operativo debe imponer esta propiedad.

Si consideramos que podría cargarse un programa ejecutable desde el disco a la memoria. Una opción consiste en cargar el programa completo en memoria física en el momento de ejecutar el programa. Sin embargo, esta técnica presenta el



problema de que puede que no necesitemos inicialmente todo el programa en la memoria. Considere un programa que comience con una lista de acciones disponibles, de entre las cuales el usuario debe seleccionar una. Cargar el programa completo en memoria hace que se cargue el código ejecutable de todas las opciones, independientemente de si el usuario selecciona o no una determinada opción. Una estrategia alternativa consiste en cargar inicialmente las páginas únicamente cuando sean necesarias. Esta técnica se denomina paginación bajo demanda y se utiliza comúnmente en los sistemas de memoria virtual. Con la memoria virtual basada en paginación bajo demanda, sólo se cargan las páginas cuando así se solicita durante la ejecución del programa; de este modo, las páginas a las que nunca se acceda no llegarán a cargarse en la memoria física.

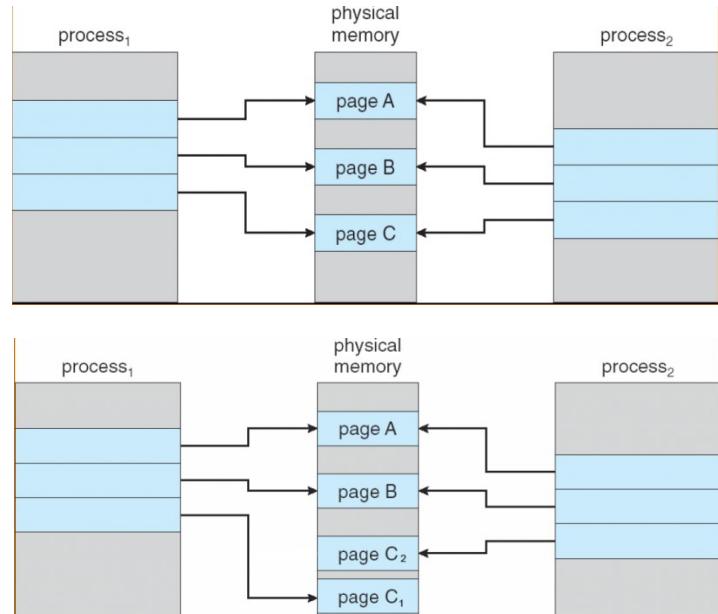
Nota: ejemplo de memoria compartida en PDF Memoria 4 diapositivas 27 y 28.

Copia en escritura: hemos indicado cómo podría un proceso comenzar rápidamente, cargando únicamente en memoria la página que contuviera la primera instrucción. Sin embargo, la creación de un proceso mediante la llamada al sistema `fork()` puede inicialmente evitar que se tenga que cargar ninguna página, utilizando una técnica similar a la de compartición de páginas. Esta técnica permite la creación rápida de procesos y minimiza el número de nuevas páginas que deben asignarse al proceso recién creado.

Recuerde que la llamada al sistema `fork()` crea un proceso hijo como duplicado de su padre. Tradicionalmente, `fork()` trabajaba creando una copia del espacio de direcciones del padre para el hijo, duplicando las páginas que pertenecen al padre. Sin embargo, considerando que muchos procesos hijos invocan la llamada al sistema `exec()` inmediatamente después de su creación, puede que sea innecesaria la copia del espacio de direcciones del padre. Como alternativa, podemos utilizar una técnica conocida con el nombre de copia durante la escritura, que funciona permitiendo que los procesos padre e hijo comparten inicialmente las mismas páginas. Estas páginas compartidas se marcan como páginas de copia durante la escritura, lo que significa que si cualquiera de los procesos escribe en una de las páginas compartidas, se creará una copia de esa página compartida. El proceso de copia durante la escritura se ilustra en las figuras, que muestran el contenido de la memoria física antes y después de que el proceso 1 modifique la página C.

Por ejemplo, si el proceso hijo trata de modificar una página que contiene parte de la pila, estando las páginas definidas como de copia durante la escritura. El sistema operativo creará entonces una copia de esta página, y la mapeará sobre el espacio de direcciones del proceso hijo. El proceso hijo modificará entonces su página copiada y no la página que pertenece al proceso padre.

Obviamente, cuando se utiliza la técnica de copia durante la escritura, sólo se copian las páginas que sean modificadas por algunos de los procesos; todas las páginas no modificadas podrán ser compartidas por los procesos padre e hijo. Observe también que sólo es necesario marcar como de copia durante la escritura aquellas páginas que puedan ser modificadas. Las páginas que no



puedan modificarse, páginas que contengan el código ejecutable, pueden compartirse entre el padre y el hijo. La técnica de copia durante la escritura resulta común en varios sistemas operativos, incluyendo Windows XP, Linux y Solaris.

Cuando se determina que se va a duplicar una página utilizando el mecanismo de copia durante la escritura, es importante fijarse en la ubicación desde la que se asignará la página libre. Muchos sistemas operativos proporcionan un conjunto compartido de páginas libres para satisfacer tales solicitudes. Esas páginas libres se suelen asignar cuando la pila o el cúmulo de un proceso deben expandirse o cuando es necesario gestionar páginas de copia durante la escritura. Los sistemas operativos asignan normalmente esas páginas utilizando una técnica denominada relleno de ceros bajo demanda. Las páginas de relleno de cero bajo demanda se llenan de ceros antes de asignarlas, eliminando así el contenido anterior.

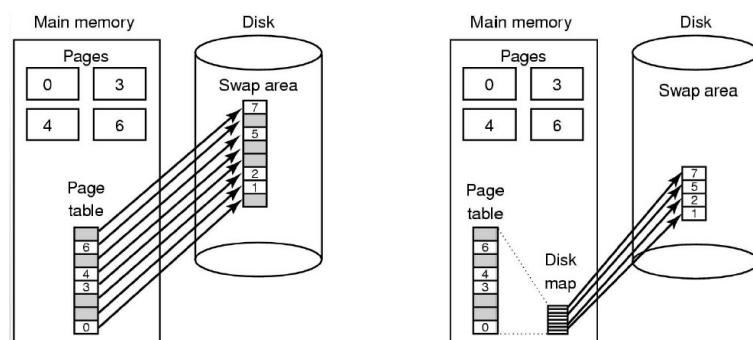
Mapeo de un archivo en memoria: una lectura secuencial de un archivo de disco utilizando las llamadas estándar al sistema `open()`, `read()` y `write()`. Cada acceso al archivo requiere una llamada al sistema y un acceso al disco. Alternativamente, podemos utilizar las técnicas de memoria virtual explicadas hasta ahora para tratar la E/S de archivo como si fueran accesos rutinarios a memoria. Esta técnica, conocida con el nombre de mapeo en memoria de un archivo, permite asociar lógicamente con el archivo una parte del espacio virtual de direcciones. El mapeo en memoria de un archivo se lleva a cabo mapeando cada bloque de disco sobre una página de la memoria. El acceso inicial al archivo se produce mediante los mecanismos ordinarios de paginación bajo demanda, provocando un fallo de página. Sin embargo, lo que se hace es leer una parte del archivo equivalente al tamaño de una página, extrayendo los datos del sistema de archivos y depositándolos en una página física, algunos sistemas pueden optar por cargar más de una página de memoria cada vez. Las subsiguientes lecturas y escrituras en el archivo se gestionarán como accesos normales de memoria, simplificando así el acceso de archivo y también su utilización, al permitir que el sistema manipule los archivos a través de la memoria en lugar de incurrir en la carga de trabajo adicional asociada a la utilización de las llamadas al sistema `read()` y `write()`.

Las escrituras en el archivo mapeado en memoria no se transforman necesariamente en escrituras inmediatas, síncronas, en el archivo del disco. Algunos sistemas pueden elegir actualizar el archivo físico cuando el sistema operativo compruebe periódicamente si la página de memoria ha sido modificada. Cuando el archivo se cierre, todos los datos mapeados en memoria se volverán a escribir en el disco y serán eliminados de la memoria virtual del proceso.

Algunos sistemas operativos proporcionan mapeo de memoria únicamente a través de una llamada al sistema específica y utilizan las llamadas al sistema normales para realizar todas las demás operaciones de E/S de archivo. Sin embargo, los otros sistemas prefieren mapear en memoria un archivo independientemente de si ese archivo se ha especificado como archivo mapeado en memoria o no.

Esta técnica es utilizada comúnmente para asociar bibliotecas compartidas.

Área de intercambio: un aspecto adicional de la paginación bajo demanda es la gestión y el uso global del espacio de intercambio. Las operaciones de E/S de disco dirigidas al espacio de intercambio son generalmente más rápidas que las correspondientes al sistema de



archivos, porque el espacio de intercambios se asigna en bloques mucho mayores y no se utilizan mecanismos de búsqueda de archivos ni métodos de asignación indirecta. El sistema puede, por tanto, conseguir 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 desde el sistema de archivos, pero ir las escribiendo en el espacio de intercambio a medida que se las sustituye. Esta técnica garantiza que sólo se lean desde el sistema de archivos las páginas necesarias y que todas las operaciones subsiguientes de paginación se lleven a cabo desde el espacio de intercambio.

Algunos sistemas tratan de 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 desde el sistema de archivos; sin embargo, cuando hace falta sustituir páginas, estos marcos pueden simplemente sobreescibirse, porque nunca se han modificado, y las páginas pueden volver a leerse desde el sistema de archivos en caso necesario. Utilizando esta técnica, el propio sistema de archivos sirve como almacén de respaldo. Sin embargo, seguirá siendo necesario utilizar espacio de intercambio para las páginas que no estén asociadas con un archivo. Estas páginas incluyen la pila y el círculo de cada proceso. Este método parece ser un buen compromiso y es el que se utiliza en varios sistemas como por ejemplo el área dedicada, separada del Sistema de Archivos en Linux o un archivo dentro del Sistema de Archivos en Windows son área de intercambio.

Nota: ejemplo de área de intercambio en PDF Memoria 4 – diapositivas 37, 38 y 39.

Dispositivos de E/S: los dispositivos que tienen que hacer E/S con las computadoras pueden clasificarse, básicamente, en tres categorías:

- II. **Legibles por los humanos:** apropiados para la comunicación con el usuario. Como ejemplo se tienen los terminales de video, que constan de un teclado, una pantalla y, quizás, otros dispositivos como un mouse o una impresora.
- III. **Legibles por la máquina:** adecuados para comunicarse con equipos electrónicos, como discos, unidades de cinta, sensores, controladores e impulsores.
- IV. **Comunicación:** apropiados para comunicarse con dispositivos lejanos. Por ejemplo adaptadores de líneas digitales y módems.

Existen grandes diferencias entre las clases de dispositivos y estas son, incluso, sustanciales, dentro de cada clase. Las siguientes son las diferencias principales:

- **Velocidad de los datos:** puede haber una diferencia de varios órdenes de magnitud en las velocidades de transmisión de datos.
- **Aplicaciones:** la utilidad que se le da a un dispositivo tiene una gran influencia en el software y en las políticas del sistema operativo y de las utilidades de apoyo. Por ejemplo, un disco que almacena archivos necesita el soporte de un software de gestión de archivos. En cambio, un disco usado como almacén de páginas de un sistema de memoria virtual dependerá del uso que se haga del hardware y el software de memoria virtual.
- **Complejidad del control:** una impresora necesita una interfaz de control relativamente simple. En cambio, un disco es mucho más complejo. El efecto de estas diferencias en el sistema operativo es filtrado, hasta cierto punto, por la complejidad del módulo de E/S que controla al dispositivo.
- **Unidad de transferencia:** los datos pueden transmitirse como flujos de bytes o caracteres, o en bloques mayores.

- **Representación de los datos:** en diferentes dispositivos se emplean diferentes esquemas de codificación de datos, incluidas las diferencias en los códigos de caracteres y los convenios de paridad.
- **Condiciones de error:** la naturaleza de los errores, la manera en que se informa sobre ellos, sus consecuencias y el rango disponible de respuestas difieren ampliamente de un dispositivo a otro.

Esta diversidad conduce hacia un enfoque consistente y uniforme de la E/S, que es difícil de alcanzar, tanto desde el punto de vista del sistema operativo como de los procesos de usuario.

A pesar de todas estas diferencias, la principal es que los módulos de entrada y salida siempre son más lentos que el CPU y la RAM.

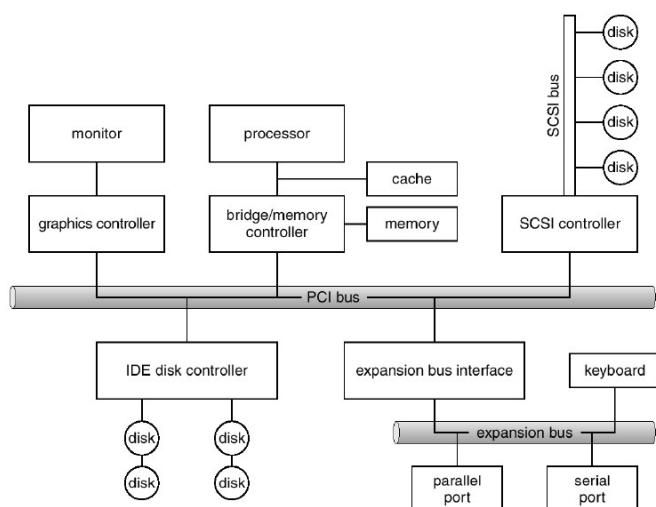
Hardware y software involucrado: a pesar de la increíble variedad de dispositivos de E/S existentes, sólo son necesarios unos cuantos conceptos para comprender cómo se conectan los dispositivos y cómo puede el software controlar el hardware asociado.

Los dispositivos se comunican con los sistemas informáticos enviando señales a través de un cable o incluso a través del aire. Cada dispositivo se comunica con la máquina a través de un punto de conexión o puerto, como por ejemplo un puerto serie. Si los dispositivos utilizan un conjunto común de hilos, dicha conexión se denomina *bus*. Un bus es un conjunto de hilos, junto con un protocolo rígidamente definido que especifica el conjunto de mensajes que pueden enviarse a través de esos hilos.

Los buses se utilizan en multitud de ocasiones dentro de la arquitectura de los sistemas informáticos. En la figura se muestra una estructura típica de un bus de PC. Esta figura muestra un bus PCI, el bus común de los sistemas PC, que conecta el subsistema procesador 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. En la parte superior derecha de la figura, podemos ver cuatro discos conectados a un bus SCSI que se inserta en una controladora SCSI.

Una controladora es una colección de componentes electrónicos que permite controlar un puerto, un bus o un dispositivo. Un ejemplo simple de controladora de dispositivo sería la controladora de un puerto serie: se trata de un único chip, o una parte de un chip, dentro de la controladora que controla las señales que se transmiten a través de los hilos de un puerto serie. Por contraste, una controladora de bus SCSI no es tan simple; como el protocolo SCSI es muy complejo, la controladora de bus SCSI se suele implementar mediante una tarjeta de circuitos separada o, adaptadora *host*, que se inserta en la computadora. Normalmente, dicha tarjeta separada contiene un procesador, microcódigo y algo de memoria privada para poder procesar los mensajes del protocolo SCSI. Algunos dispositivos tienen sus propias controladoras integradas.

La controladora dispone de uno o más registros para los datos y las señales de control. El procesador se comunica con la controladora leyendo y escribiendo patrones de bits en dichos registros. Una forma de llevar a cabo esta comunicación es utilizando instrucciones de E/S especiales que especifican la transferencia de un byte o de una palabra a una dirección de puerto de E/S. La instrucción de E/S configura las líneas de bus para seleccionar el dispositivo apropiado y



para leer o escribir bits en un registro del dispositivo. Alternativamente, la controladora de dispositivo puede soportar una E/S mapeada en memoria. En este caso, los registros de control del dispositivo están mapeados en el espacio de direcciones del procesador. La CPU ejecuta las solicitudes utilizando instrucciones estándar de transferencia de datos para leer y escribir los registros de control del dispositivo.

Algunos sistemas utilizan ambas técnicas. Por ejemplo, un PC utiliza instrucciones de E/S para controlar algunos dispositivos y E/S mapeada en memoria para controlar otros. Por ejemplo, una controladora gráfica tiene puertos de E/S para las operaciones básicas de control, pero también dispone de una gran región mapeada en memoria para almacenar el contenido de la pantalla. El proceso envía la salida a la pantalla escribiendo datos en esa región mapeada en memoria. La controladora genera la imagen de pantalla basándose en el contenido de esa memoria. Esta técnica resulta muy simple de utilizar; además, resulta más rápido escribir millones de bytes en la memoria gráfica que ejecutar millones de instrucciones de E/S. Pero la facilidad de escritura en una controladora de E/S mapeada en memoria se ve compensada por una desventaja; puesto que uno de los tipos más comunes de fallo de software es la escritura mediante un puntero incorrecto en una región de memoria distinta de la que se pretendía, los registros de dispositivos mapeados en memoria son vulnerables a la modificación accidental por parte de los programas. Por supuesto, la memoria protegida nos ayuda a reducir este riesgo.

Un puerto de E/S está compuesto típicamente de cuatro registros, denominados:

- I. Registro de estado.
- II. Registro de control.
- III. Registro de entrada de datos.
- IV. Registro de salida de datos.

Los drivers son actores fundamentales en los dispositivos de E/S, ya que son aquellos que le indican al sistema operativo, cómo manejar tales dispositivos.

Técnica de comunicación de E/S: para las operaciones de E/S son posibles las 3 técnicas siguientes:

- I. **E/S programada, polling:** cuando el procesador está ejecutando un programa y encuentra una instrucción de E/S, ejecuta dicha instrucción, enviando una orden al módulo apropiado de E/S. Con E/S programada, el módulo de E/S llevará a cabo la acción requerida y luego activará los bits apropiados en el registro de estado de E/S. El módulo de E/S no lleva a cabo ninguna otra acción para avisar al procesador. En particular, no interrumpe al procesador. Así pues, es responsabilidad del procesador comprobar periódicamente el estado del módulo de E/S hasta saber que se ha completado la operación.

Con esta técnica, el procesador es el responsable de extraer los datos de la memoria principal cuando va a hacer una salida o poner los datos en la memoria principal cuando se hace una entrada. El software de E/S se escribe de manera tal que el procesador ejecute unas instrucciones que le otorguen el control directo sobre la operación de E/S, incluyendo la comprobación del estado de los dispositivos, el envío de órdenes de lectura o escritura y la transferencia de los datos. Por lo tanto, en el conjunto de instrucciones se incluyen instrucciones de E/S de las siguientes categorías:

- **Control:** empleadas para activar un dispositivo externo y decirle qué debe hacer. Por ejemplo, una unidad de cinta magnética puede ser instruida para rebobinar o avanzar un registro.
- **Comprobación:** empleadas para comprobar ciertas condiciones de estado asociadas con un módulo de E/S y sus periféricos.

- **Lectura, escritura:** empleadas para transferir los datos entre los registros del procesador y los dispositivos.
- II. **E/S dirigida por interrupciones:** el problema de la E/S programada es que el procesador tiene que esperar un largo rato a que el módulo de E/S en cuestión esté listo para recibir o transmitir más datos. El procesador, mientras está esperando, debe interrogar repetidamente el estado del módulo de E/S. Como resultado, el nivel de rendimiento del sistema en conjunto se degrada fuertemente.
- Una alternativa es que el procesador envíe una orden de E/S al módulo y se dedique a hacer alguna otra tarea útil. El módulo de E/S interrumpirá entonces al procesador para requerir sus servicios cuando esté listo para intercambiar los datos. El procesador ejecuta entonces la transferencia de los datos y reanuda el procesamiento anterior.
- Seguidamente veremos cómo funciona esto, en primer lugar desde el punto de vista del módulo de E/S. Para la entrada, el módulo de E/S recibe una orden LEER desde el procesador. El módulo de E/S procede entonces con la lectura de los datos desde el periférico asociado. Una vez que los datos están en el registro de datos del módulo, este envía una señal de interrupción al procesador a través de una línea de control. El módulo espera entonces a que los datos sean solicitados por el procesador. Cuando se haga esta solicitud, el módulo pondrá los datos en el bus de datos y estará listo para otra operación de E/S.
- Desde el punto de vista del procesador, la acción para la entrada es como sigue. El procesador envía una orden LEER. Salva entonces el contexto, el contador de programa y los registros del procesador, del programa en curso, se sale del mismo y se dedica a hacer otra cosa, por ejemplo, el procesador puede estar trabajando con diferentes programas al mismo tiempo. Al finalizar cada ciclo de instrucción, el procesador comprueba si hubo alguna interrupción. Cuando se produce una interrupción desde el módulo de E/S, el procesador salva el contexto del programa que está ejecutando en ese momento y comienza la ejecución de la rutina de tratamiento de la interrupción. En este caso, el procesador lee la palabra de datos del módulo de E/S y la almacena en la memoria. Luego restaura el contexto del programa que emitió la orden de E/S o de algún otro programa y reanuda la ejecución.
- Sin embargo, la E/S por interrupciones sigue consumiendo una gran cantidad de tiempo del procesador, debido a que cada palabra de datos que va de la memoria al módulo de E/S o del módulo de E/S a la memoria debe pasar a través del procesador.
- III. **Acceso directo a memoria, DMA:** la E/S dirigida por interrupciones, aunque es más eficiente que la simple E/S programada, todavía requiere de la intervención activa del procesador para transferir los datos entre la memoria y un módulo de E/S y, además, cualquier transferencia de datos debe recorrer un camino que pasa por el procesador. Así pues, ambas formas de E/S adolecen de dos desventajas inherentes:
- La velocidad de transferencia de E/S está limitada por la velocidad con la que el procesador puede comprobar y dar servicio a un dispositivo.
 - El procesador participa en la gestión de la transferencia de E/S; debe ejecutarse una serie de instrucciones en cada transferencia de E/S.
- Cuando se tienen que mover grandes volúmenes de datos, se necesita una técnica más eficiente: el acceso directo a memoria, DMA, *Direct Memory Access*. La función de DMA se puede llevar a cabo por medio de un módulo separado sobre el bus del sistema o puede estar incorporada dentro de un módulo de E/S. En cualquier caso, la técnica funciona como sigue. Cuando el procesador desea leer o escribir un bloque de datos, emite una orden hacia el módulo de DMA, enviándole la información siguiente:

- Si lo que se solicita es una lectura o una escritura.
- La dirección del dispositivo de E/S involucrado.
- La dirección inicial de memoria desde la que se va a leer o a la que se va a escribir.
- El número de palabras a leer o escribir.

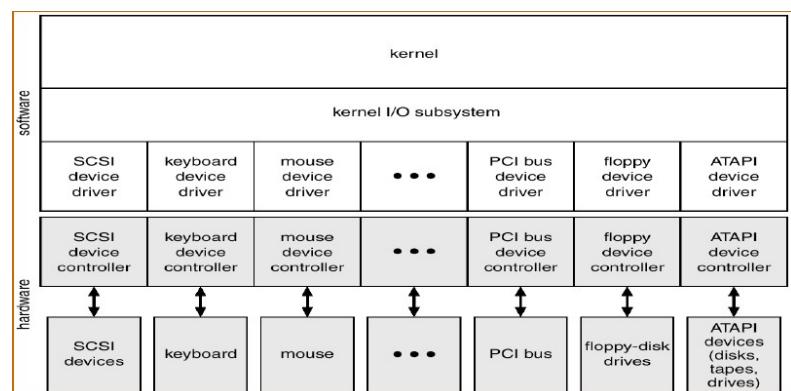
El procesador continúa entonces con otro trabajo. Habrá delegado la operación de E/S en el módulo de DMA y dicho módulo es el que tendrá que encargarse de esta. El módulo de DMA transfiere el bloque entero, una palabra cada vez, directamente hacia o desde la memoria, sin pasar por el procesador. Cuando se completa la transferencia, el módulo de DMA envía una señal de interrupción al procesador. De esta manera, el procesador se ve involucrado solo al inicio y al final de la transferencia.

El módulo de DMA debe tomar el control del bus para transferir los datos con la memoria. Debido a la competencia por la utilización del bus, puede ocurrir que el procesador necesite el bus pero deba esperar. Nótese que esto no es una interrupción; el procesador no salva el contexto y se dedica a hacer otra cosa. En su lugar, el procesador hace una pausa durante un ciclo del bus. El efecto general es hacer que el procesador ejecute con más lentitud durante una transferencia de DMA. No obstante, para una transferencia de E/S de varias palabras, el DMA es bastante más eficiente que la E/S programada o la dirigida por interrupciones.

Cuando el módulo de E/S en cuestión es un sofisticado canal de E/S, el concepto de DMA debe tenerse en cuenta con más razón. Un canal de E/S es un procesador propiamente dicho, con un conjunto especializado de instrucciones, diseñadas para la E/S. En un sistema informático con tales dispositivos el procesador no ejecuta instrucciones de E/S. Dichas instrucciones se almacenan en la memoria principal para ser ejecutadas por el propio canal de E/S. Así pues, el procesador inicia una transferencia de E/S instruyendo al canal de E/S para ejecutar un programa en memoria. El programa especifica el o los dispositivos, la zona o zonas de memoria para almacenamiento, la prioridad y la acción que llevar a cabo bajo ciertas condiciones de error. El canal de E/S sigue estas instrucciones y controla la transferencia de datos, informando de nuevo al procesador al terminar.

Interfaz de E/S: son las técnicas de estructuración y de las interfaces del sistema operativo que permiten tratar de una forma estándar y uniforme a los dispositivos de E/S.

Al igual que otros problemas complejos de ingeniería del software, la técnica utilizada aquí se basa en los conceptos de abstracción, encapsulación y descomposición del software en niveles. Específicamente, podemos abstraer las diferencias de detalle existentes entre los dispositivos de E/S identificando unos cuantos tipos generales de dispositivo. Para acceder a cada tipo general se utiliza un conjunto estandarizado de funciones, es decir, una interfaz. Las diferencias se encapsulan en módulos del *kernel* denominados controladores del dispositivo que están personalizados internamente para cada dispositivo pero exportan una de las interfaces estándar. La figura ilustra cómo se estructuran en niveles de software las partes de *kernel* relacionadas con la E/S.



El propósito de la capa de controladores de dispositivo es ocultar a ojos del subsistema de E/S del *kernel* las diferencias existentes entre las controladoras de dispositivo, de forma similar a como las llamadas al sistema de E/S encapsulan el comportamiento de los dispositivos en unas cuantas clases genéricas que ocultan a ojos de las aplicaciones las diferencias hardware. Hacer el subsistema de E/S independiente del hardware simplifica el trabajo del desarrollador de sistemas operativos y también beneficia a los fabricantes de hardware. Éstos pueden diseñar los nuevos dispositivos para que sean compatibles con una interfaz de controladora *host* existente o pueden escribir controladores de dispositivo para implementar la interfaz del nuevo hardware con una serie de sistemas operativos populares. De este modo, podemos conectar nuevos periféricos a una computadora sin esperar a que el fabricante del sistema operativo desarrolle el correspondiente código de soporte.

Desafortunadamente para los fabricantes de dispositivos hardware, cada tipo de sistema operativo tiene sus propios estándares en cuanto a la interfaz del controlador de dispositivo. Un dispositivo determinado puede comercializarse junto con múltiples controladores de dispositivo, como por ejemplo controladores para MS-DOS, Windows 95/98, Windows N T / 2000 v Solaris. Los dispositivos pueden variar desde muchos puntos de vista, como se ilustra en la figura.

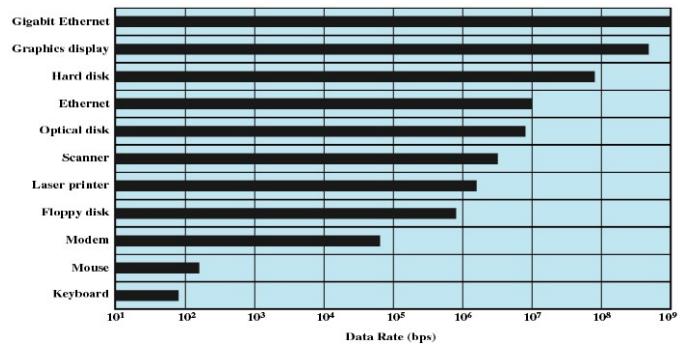
aspect	variation	example
data-transfer mode	character block	terminal disk
access method	sequential random	modem CD-ROM
transfer schedule	synchronous asynchronous	tape keyboard
sharing	dedicated sharable	tape keyboard
device speed	latency seek time transfer rate delay between operations	
I/O direction	read only write only read&write	CD-ROM graphics controller disk

- **Flujo de caracteres o bloque:** un dispositivo de flujo de caracteres transfiere los bytes uno a uno, mientras que un dispositivo de bloque transfiere un bloque de bytes como una sola unidad.
 - **Flujo de bloques:** captura todos los aspectos necesarios para acceder a unidades de disco y a otros dispositivos orientados a bloques. El dispositivo debe comprender comandos tales como `read()` o `Write()`, si se trata de un dispositivo de acceso aleatorio, también se espera que disponga de un comando `seek()` para especificar qué bloque hay que transferir a continuación. Las aplicaciones acceden normalmente a este tipo de dispositivos mediante una interfaz de sistema de archivos. Podemos ver que `read()`, `write()` y `seek()` capturan el comportamiento esencial de los dispositivos de almacenamiento de bloques, de modo que las aplicaciones quedan aisladas de las diferencias de bajo nivel que puedan existir entre los dispositivos.
 - **Flujo de caracteres:** las llamadas básicas al sistema en esta interfaz permiten a las aplicaciones leer o escribir un carácter, mediante las llamadas `get()` y `put()`. Por encima de esta interfaz, pueden construirse bibliotecas que permitan el acceso de línea en línea, con servicios de almacenamiento en búfer y edición.
- **Accesos secuencial o aleatorio:** un dispositivo secuencial transfiere los datos en un orden fijo determinado por el dispositivo, mientras que el usuario de un dispositivo de acceso aleatorio puede instruir al dispositivo para que se posicione en cualquiera de las ubicaciones disponibles de almacenamiento de datos.
- **Síncrono o asíncrono:** un dispositivo síncrono realiza transferencias de datos con tiempos de respuesta predecibles. Un dispositivo asíncrono exhibe unos tiempos de respuesta irregulares o no predecibles.

- **Compatible o dedicado:** un dispositivo compatible puede ser usado de forma concurrente por varios procesos o hebras, por ejemplo un disco rígido; un dispositivo dedicado no puede ser compartido de esta forma, por ejemplo una impresora.
- **Velocidad de operación:** las velocidades de los dispositivos van desde unos pocos bytes por segundo a unos cuantos gigabytes por segundos.
- **Lectura-escritura, sólo lectura o sólo escritura:** algunos dispositivos realizan tanto entrada como salida, por ejemplo un disco, pero otros sólo soportan una única dirección de transferencia de los datos. Por ejemplo las pantallas solo soportan escrituras, mientras que los CD-ROM soportan solo lectura.

En lo que respecta al acceso por parte de las aplicaciones, muchas de estas diferencias quedan ocultas gracias al sistema operativo, y los dispositivos se agrupan en unos cuantos tipos convencionales. Los estilos resultantes de acceso al dispositivo han demostrado ser en la práctica muy útil y de aplicación general. Aunque las llamadas exactas al sistema pueden diferir de unos sistemas operativos a otros, las categorías de dispositivo son bastante estándar. Los principales sistemas de acceso incluyen la E/S de bloque, la E/S de flujo de caracteres, el acceso a archivos mapeados en memoria y los *sockets* de red. Los sistemas operativos también proporcionan llamadas especiales al sistema para acceder a unos cuantos dispositivos adicionales, como por ejemplo un reloj o un contador. Algunos sistemas operativos proporcionan un conjunto de llamadas al sistema para dispositivos de visualización gráfica, de vídeo y de audio.

Uno de los conflictos más graves entre las E/S y las computadoras, es que las velocidades de procesamiento de ambos son totalmente diferentes. Las computadoras tienen una velocidad de procesamiento de datos mucho mayor que al de los dispositivos de E/S. La necesidad de este desacoplamiento se ilustra en la figura, donde se indican las enormes diferencias en velocidades de los dispositivos para un hardware típico de una computadora.



Subsistema de E/S: un *kernel* proporciona muchos servicios relacionados con la E/S. Varios de los servicios de planificación, almacenamiento en búfer, almacenamiento en caché, gestión de colas, reserva de dispositivos y tratamiento de errores, son proporcionados por el subsistema de E/S del *kernel* y se construyen sobre la infraestructura formada por el hardware y los controladores de dispositivo. El subsistema de E/S también es responsable de protegerse a sí mismo de los procesos erróneos y de los usuarios maliciosos.

- **Planificación de E/S:** planificar un conjunto de solicitudes de E/S significa determinar un orden adecuado en el que ejecutarlas. El orden en el que las aplicaciones realizan las llamadas al sistema no suele ser la mejor elección. La planificación puede mejorar el rendimiento global del sistema, permite compartir el acceso a los dispositivos equitativamente entre los distintos procesos y puede reducir el tiempo de espera medio requerido para que la E/S se complete. Un ejemplo simple para ilustrar estos aspectos: suponga que el brazo de un disco está situado cerca del principio del mismo y que tres aplicaciones distintas ejecutan llamadas de lectura bloqueantes dirigidas a dicho disco. La aplicación 1 solicita un bloque situado cerca del final del disco, la aplicación 2 solicita otro situado cerca del principio y la aplicación 3 solicita un bloque en parte intermedia del disco. El sistema operativo puede reducir la distancia recorrida por el brazo del disco

sirviendo a las aplicaciones en el orden 2, 3, 1. Reordenar el servicio de las distintas solicitudes de esta manera es la esencia de los mecanismos de planificación de E/S.

- **Buferring:** es un área de memoria que almacena datos mientras se están transfiriendo entre dos dispositivos o entre un dispositivo y una aplicación. El almacenamiento en búfer se realiza por tres razones. Una razón es realizar una adaptación de velocidades entre el productor v el consumidor de un flujo de datos. Suponga, por ejemplo, que se está recibiendo un archivo vía módem para su almacenamiento en el disco duro. El módem es unas 1000 veces más lento que el disco duro, por lo que se crea un búfer en memoria principal para acumular los bytes recibidos del módem. Una vez que ha llegado un búfer completo de datos, puede escribirse el búfer en disco en una sola operación. Puesto que la escritura en el disco no es instantánea y el módem sigue necesitando un lugar para almacenar los datos adicionales entrantes, se utilizan los búferes. Después de que el módem llene el primer búfer, se solicita la escritura en disco v el módem comienza entonces a llenar el segundo búfer mientras que el primero se escribe en el disco. Para cuando el módem haya llenado el segundo búfer, la escritura en disco del primero deberá haberse completado, por lo que el módem podrá conmutar de nuevo al primer búfer mientras se escribe en el disco el contenido del segundo. Esta técnica de doble búfer desacopla al productor de datos del consumidor de los mismos, relajando así los requisitos de temporización entre ellos.
- **Caching:** es una región de memoria rápida que alberga copias de ciertos datos. El acceso a una copia almacenada en caché es más eficiente que el acceso al original. Por ejemplo las instrucciones del proceso actualmente en ejecución están almacenadas en el disco, almacenadas en la caché de la memoria física y almacenadas también en las cachés principal y secundaria de la CPU. La diferencia entre un búfer y una caché es que un búfer puede almacenar la única copia existente de un elemento de datos, mientras que una caché, por definición, almacena simplemente en un dispositivo de almacenamiento más rápido una copia de un elemento que reside en algún otro lugar.
El almacenamiento en caché y el almacenamiento en búfer son funciones bien diferenciadas, aunque en ocasiones puede utilizarse una misma región de memoria para ambos propósitos. Por ejemplo, para preservar la semántica de copia y para permitir una eficiente planificación de la E/S de disco, el sistema operativo utiliza búferes en memoria principal para almacenar los datos del disco. Estos búferes también se emplean como caché, para mejorar la eficiencia de E/S para aquellos archivos que estén compartidos por varias aplicaciones o que estén siendo escritos y vuelto a leer de forma rápida. Cuando el *kernel* recibe una solicitud de E/S de archivo, accede primero a la caché de búfer para ver si dicha región del archivo ya está disponible en la memoria principal. En caso afirmativo, podemos editar o diferir una E/S de disco físico. Asimismo, las escrituras en disco se acumulan en la caché del búfer durante varios segundos, con el fin de componer grandes transferencias de datos y permitir así una planificación eficiente de las operaciones de escritura.
- **Spooling:** es un búfer que almacena la salida dirigida a un dispositivo, como por ejemplo una impresora, que no pueda aceptar flujos de datos entrelazados. Aunque una impresora sólo puede dar servicio a un cierto trabajo cada vez, es posible que varias aplicaciones quieran imprimir sus datos de salida de manera concurrente, sin que la salida de unas aplicaciones se mezcle con la de otras. El sistema operativo resuelve este problema interceptando toda la salida dirigida a la impresora. La salida de cada aplicación se almacena temporalmente en un archivo de disco separado. Cuando una aplicación termina de imprimir, el sistema de gestión de colas pone el correspondiente archivo

temporal en la cola de salida de la impresora. El sistema de gestión de colas va copiando los archivos de salida en la impresora de uno en uno. En algunos sistemas operativos, estas colas de impresión se gestionan mediante un proceso demonio del sistema. En otros, se gestiona mediante una hebra interna al *kernel*. En cualquiera de los dos casos el sistema operativo proporciona una interfaz de control que permite a los usuarios y a los administradores del sistema visualizar la cola de solicitudes de impresión, eliminar los trabajos no deseados antes de que lleguen a imprimirse, suspender la impresión mientras se está solventando algún error de impresora, etc.

Algunos dispositivos, como las unidades de cinta y las impresoras, no pueden multiplexar las solicitudes de E/S de múltiples aplicaciones concurrentes. La gestión de colas de impresión es una de las formas en que el sistema operativo puede coordinar estas operaciones concurrentes de salida. Otra forma de tratar con el acceso concurrente a los dispositivos consiste en proporcionar facilidades explícitas de coordinación. Algunos sistemas operativos, incluyendo VMS, proporcionan soporte para el acceso exclusivo a los dispositivos, permitiendo a los procesos asignar un dispositivo inactivo y desecharlo cuando ya no sea necesario. Otros sistemas operativos imponen un límite de un sólo descriptor de archivo abierto para tales dispositivos. Muchos sistemas operativos proporcionan funciones que permiten a los procesos coordinar entre sí el acceso exclusivo. Por ejemplo, Windows NT proporciona llamadas al sistema para realizar una espera hasta que un objeto dispositivo pase a estar disponible. También dispone de un parámetro de la llamada al sistema `open()` que declara los tipos de acceso que deben permitirse a otras hebras concurrentes. En estos sistemas, es responsabilidad de las aplicaciones evitar los interbloqueos.

- **Tratamiento de errores:** un sistema operativo que utilice memoria protegida puede defenderse frente a muchos tipos de errores hardware y de las aplicaciones, de modo que cada pequeño error no provoque un fallo completo del sistema. Los dispositivos y las transferencias de E/S pueden fallar de muchas formas, debido a razones transitorias, como por ejemplo cuando una red se sobrecarga, o a razones "permanentes", como por ejemplo si falla una controladora de disco. Los sistemas operativos pueden a menudo compensar de manera efectiva los fallos transitorios. Por ejemplo, un fallo de una operación `read()` de disco provoca que se reintente la operación `read()` y un error en una operación `send()` a través de la red provoca una operación de reenvío `resend()`, si el protocolo así lo especifica. Desafortunadamente, si un componente importante experimenta un fallo de carácter permanente, resulta poco probable que el sistema operativo pueda recuperarse.

Como regla general, una llamada de E/S al sistema devolverá un bit de información acerca del estado de la llamada, mediante el que se indicará si ésta ha tenido éxito o no. En el sistema operativo UNIX, se utiliza una variable entera adicional denominada `errno` para devolver un código de error, de entre un centenar de valores posibles, que indica la naturaleza general del falle, por ejemplo, argumento fuera de rango, puntero erróneo o archivo no abierto. Por contraste, ciertos tipos de hardware pueden proporcionar información de error altamente detallada, aunque muchos de los sistemas operativos actuales no están diseñados para transmitir esta información a la aplicación. Por ejemplo, un fallo en un dispositivo SCSI se documenta por parte del protocolo SCSI en tres niveles de detalle: una clave de tipo que identifica la naturaleza general del fallo, como por ejemplo un error hardware o una solicitud ilegal; un código de tipo adicional que indica la categoría del fallo, como por ejemplo un parámetro erróneo de comando o un fallo de auto-test; y un cualificador adicional del código que proporciona todavía más detalle,

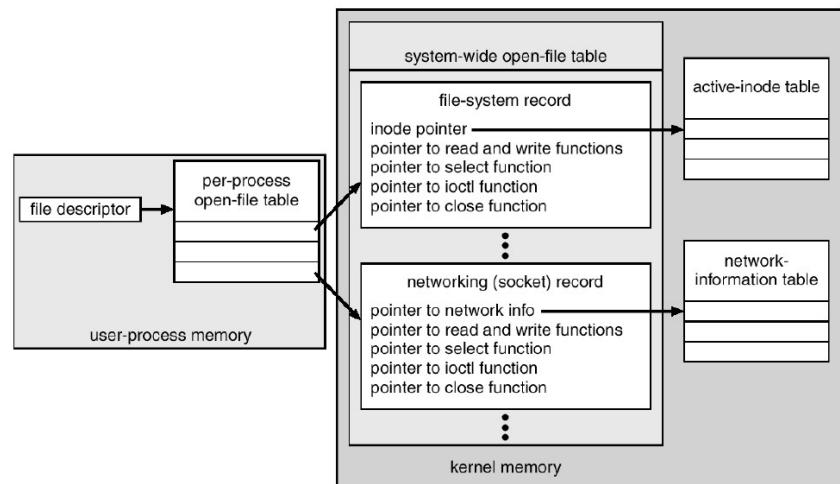
como por ejemplo qué parámetro del comando era erróneo o qué subsistema hardware ha fallado al hacer el auto-test. Además, muchos dispositivos SCSI mantienen páginas internas de información de registro de errores que pueden ser solicitadas por el *host*, aunque raramente se solicitan.

- **Protección de E/S:** los errores están estrechamente relacionados con las cuestiones de protección. Un proceso de usuario puede intentar accidental o deliberadamente interrumpir la operación normal de un sistema, tratando de ejecutar instrucciones de E/S ilegales. Podemos utilizar varios mecanismos para garantizar que ese tipo de problemas no aparezcan en el sistema.

Para evitar que los usuarios realicen operaciones de E/S ilegales, definimos todas las instrucciones de E/S como instrucciones privilegiadas. Así, los usuarios no pueden ejecutar instrucciones de E/S directamente, sino que tienen que hacerlo a través del sistema operativo. Para llevar a cabo una operación de E/S, el programa de usuario ejecuta una llamada al sistema para solicitar que el sistema operativo realice esa operación de E/S por cuenta suya. El sistema operativo, ejecutándose en modo monitor, comprueba que la solicitud es válida y, en caso afirmativo realiza la E/S solicitada. A continuación, el sistema operativo devuelve el control al usuario.

Además, habrá que utilizar el sistema de protección de memoria para proteger todas las ubicaciones de memoria mapeada y de los puertos de E/S frente a los accesos de los usuarios. El *kernel* no puede simplemente denegar todos los accesos de los usuarios. La mayoría de los juegos gráficos y del software de edición y reproducción de vídeo necesitan acceso directo a la memoria de la controladora gráfica mapeada en memoria, con el fin de acelerar la velocidad de los gráficos. El *kernel* puede en este caso proporcionar un mecanismo de bloqueo para permitir que se asigne a un proceso cada vez una cierta sección de la memoria gráfica.

- **Estructuras de datos del kernel:** el *kernel* necesita mantener información de estado acerca del uso de los componentes de E/S. El *kernel* hace esto mediante diversas estructuras de datos internas al *kernel*. El *kernel* utiliza muchas estructuras similares para controlar las conexiones de red, las comunicaciones con los dispositivos de tipo carácter y otras actividades de E/S.



UNIX proporciona acceso de sistema de archivos a diversas entidades, como los archivos de usuario, los dispositivos sin formato y los espacios de direcciones de los procesos.

Aunque cada una de estas actividades soporta una operación `read()`, la semántica difiere. Por ejemplo, para leer un archivo de usuario, el *kernel* necesita comprobar la caché de búfer antes de decidir si debe realizar una E/S de disco. Para leer un disco sin formato, el *kernel* necesita garantizar que el tamaño de la solicitud sea un múltiplo del tamaño del sector de disco y que esté alineada con una frontera de sector. Para leer la imagen de un proceso, simplemente basta con copiar los datos desde memoria. UNIX

encapsula estas diferencias dentro de una estructura uniforme utilizando una técnica de orientación a objetos. El registro de archivos abiertos, mostrado en la figura, contiene una tabla de despacho que almacena punteros a las rutinas apropiadas, dependiendo del tipo de archivo.

Algunos sistemas operativos utilizan métodos de orientación a objetos de forma todavía más intensiva. Por ejemplo, Windows NT utiliza una implementación de paso de mensajes para la E/S. Cada solicitud de E/S se convierte en un mensaje que se envía a través del *kernel* al gestor de E/S y luego al controlador de dispositivo, cada uno de los cuales puede modificar el contenido del mensaje. Para las operaciones de salida, el mensaje contiene los datos que hay que escribir. Para las operaciones de entrada, el mensaje contiene un búfer para recibir los datos. La técnica de paso de mensajes puede añadir carea de proceso adicional, por comparación con las técnicas procedimentales que utilizan estructuras de datos compartidas, pero simplifica la estructura y el diseño del sistema de E/S y añade un grado considerable de flexibilidad.

Transformaciones de las solicitudes de E/S en operaciones hardware: hemos descrito el procedimiento de negociación entre un controlador de dispositivo y una tarjeta controladora de dispositivo, pero no hemos explicado cómo conecta el sistema operativo una solicitud de aplicación con un conjunto de hilos- de red o con un sector de disco específico. Vamos a considerar el ejemplo de la lectura de un archivó de disco. La aplicación hace referencia a los datos mediante el nombre del archivo. Dentro de un disco, el sistema de archivos mapea los nombres de archivo mediante una serie de directorios para averiguar la asignación de espacio del archivo. Por ejemplo, en MS-DOS, el nombre se mapea sobre un número que indica una entrada dentro de la tabla de acceso a los archivos y dicha entrada de la tabla nos dice qué bloques de disco están asignados al archivo. En UNIX, el nombre se mapea sobre un número del nodo, y el correspondiente al nodo contiene la información de asignación de espacio. La primera parte de un nombre de archivo MS-DOS, situado delante del carácter de dos puntos, es una cadena que identifica un dispositivo hardware específico. Por ejemplo, *c:* es la primera parte de todos los nombres de archivo que hagan referencia al disco duro principal. El hecho de que *c:* represente el disco duro principal está pre-escrito dentro del sistema operativo; *c:* se mapea sobre una dirección de puerto específica a través de una tabla de dispositivos. Debido al separador de los dos puntos, el espacio de nombres de dispositivos está separado del espacio de nombres del sistema de archivos dentro de cada dispositivo. Esta separación hace que resulte sencillo para el sistema operativo asociar una funcionalidad adicional con cada dispositivo. Por ejemplo, resulta fácil invocar los mecanismos de gestión de colas de impresión para los archivos que se escriban en la impresora.

Si, en lugar de ello, el espacio de nombres de los dispositivos está incorporado dentro del espacio de nombres normal del sistema de archivos, como es el caso en UNIX, los servicios de nombres normales del sistema de archivos se proporcionan de manera automática. Si el sistema de archivos proporciona mecanismos de control de propiedad y de control de acceso para todos los nombres de archivo, entonces los dispositivos tendrán propietarios y mecanismos de control de acceso. Puesto que los archivos se almacenan en dispositivos, dicha interfaz proporciona acceso al sistema de E/S en dos niveles: los nombres pueden utilizarse para acceder a los propios dispositivos o para acceder a los archivos almacenados en los dispositivos.

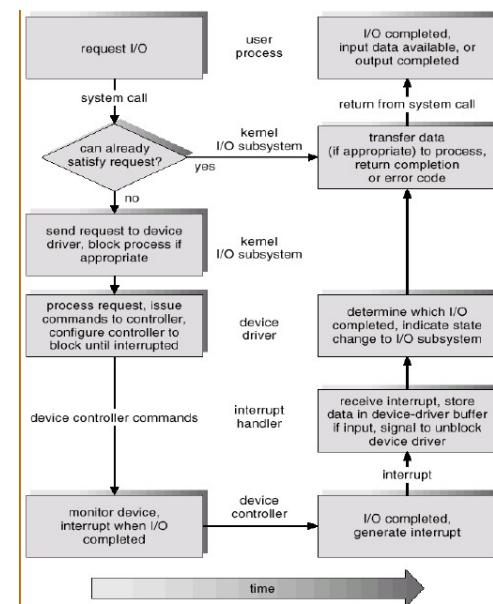
UNIX representa los nombres de los dispositivos dentro del espacio de nombres normal del sistema de archivos. A diferencia de un nombre de archivo MS-DOS, que tiene un separador de dos puntos, un nombre de ruta UNIX no tiene una separación clara de la parte correspondiente al dispositivo. De hecho, ninguna parte concreta del nombre de ruta es el nombre de un dispositivo.

UNIX tiene una tabla de montaje que asocia prefijos de los nombres de ruta con nombres de dispositivo específicos. Para resolver el nombre de ruta, UNIX busca el nombre en la tabla de montaje para localizar el prefijo correspondiente de mayor longitud; la entrada correspondiente de la tabla de montaje proporcionará el nombre de dispositivo. Este nombre de dispositivo también tiene la forma de un nombre dentro del espacio de nombres del sistema de archivos. Cuando UNIX busca este nombre en las estructuras de directorio del sistema de archivos, lo que encuentra no es un número del nodo, sino un número de dispositivo *<principal, secundario>*. El número de dispositivo principal identifica un controlador de dispositivo al que habrá que llamar para tratar las operaciones de E/S dirigidas a este dispositivo. El número de dispositivo secundario se pasa al controlador de dispositivo como índice para una tabla de dispositivos. La entrada correspondiente de la tabla de dispositivos proporciona la dirección de puerto o la dirección mapeada en memoria del controlador de dispositivo.

Los sistemas operativos modernos obtienen una gran flexibilidad de estas múltiples etapas de tablas de búsqueda dentro de la ruta comprendida entre una solicitud y una controladora física de dispositivo. Los mecanismos que pasan las solicitudes entre las aplicaciones y los controladores son generales, por lo que podemos introducir nuevos dispositivos y controladores en una computadora sin necesidad de recompilar el *kernel*. De hecho, algunos sistemas operativos tienen la capacidad de cargar controladores de dispositivo bajo demanda. En el momento del arranque, el sistema comprueba los buses hardware para determinar qué dispositivos están presentes y luego carga los controladores necesarios, inmediatamente o cuando sean requeridos por primera vez por una solicitud de E/S.

El ciclo típico de vida de una solicitud de lectura bloqueante, como se muestra en la figura. La figura sugiere que una operación de E/S requiere muchos pasos distintos que consumen, entre todos, un número enorme de ciclos de CPU:

- I. Un proceso ejecuta una llamada al sistema bloqueante `read()` dirigida a un descriptor de archivo o a un archivo que se hay abierto anteriormente.
- II. El código de la llamada al sistema en el *kernel* comprueba la corrección de los parámetros. En el caso de una operación de entrada, si los datos ya están disponibles en la caché de búfer, los datos se devuelven en el proceso y se completa la solicitud de E/S.
- III. En caso contrario, es necesario realizar una E/S física. El proceso se elimina de la cola de ejecución y se coloca en la cola de espera en el dispositivo, planificándose la solicitud de E/S. Eventualmente, el subsistema de E/S enviará la solicitud al controlador de dispositivo. Dependiendo del sistema operativo, la solicitud se envía mediante una llamada a subrutina o un mensaje interno al *kernel*.
- IV. El controlador de dispositivo asigna espacio de búfer del *kernel* para recibir los datos y planifica la E/S. Eventualmente, el controlador envía los comandos a la tarjeta controladora de dispositivo escribiendo en los registros de control del dispositivo.
- V. La controladora del dispositivo opera el hardware del dispositivo para realizar la transferencia de datos.
- VI. El controlador puede realizar un sondeo para ver la información de estado y recolectar los datos, o puede haber configurado una transferencia de DMA hacia la memoria del *kernel*.



- Estamos asumiendo que la transferencia es gestionada por una controladora de DMA, que generará una interrupción cuando la transferencia se complete.
- VII. La rutina correcta de tratamiento de interrupciones recibirá la interrupción a través de la tabla de vectores de interrupción, almacenará los datos necesarios, efectuará una señalización dirigida al controlador de dispositivo y volverá de la interrupción.
 - VIII. El controlador de dispositivo recibe la señal, determina qué solicitud de E/S se ha completado, determina el estado de la solicitud y señaliza al subsistema de E/S del *kernel* que la solicitud se ha completado.
 - IX. El *kernel* transfiere los datos a los códigos de retomo al espacio de direcciones del proceso solicitante y mueve el proceso de la cola de espera a la cola de procesos preparados.
 - X. Al mover el proceso a la cola de procesos preparados se desbloquea el proceso. Cuando el planificador asigne la CPU al proceso, éste reanudará su ejecución en el punto correspondiente a la terminación de la llamada al sistema.

Drivers: es un pequeño software que conecta el sistema operativo directamente con los componentes del hardware de la PC. Por ejemplo, una placa de vídeo instalada en tu computadora, esta necesita entenderse con el sistema operativo para poder recibir las instrucciones y procesar todo correctamente; y es justamente esto lo que hace el driver, un puente entre ambos. El driver le da instrucciones al sistema operativo, sobre cómo debe funcionar determinado hardware y de que forma el sistema debe trabajar en conjunto para suministrarte los mejores resultados.

Cada sistema operativo usa drivers diferentes y es ahí donde surgen los problemas, ya que los fabricantes raramente crean un archivo universal para todos los sistemas y el que acaba teniendo problemas buscando drivers apropiados es el usuario, y el más perjudicado cuando no se encuentran los drivers en la página del fabricante.

Esos problemas generalmente ocurren debido a la incompatibilidad entre los sistemas operativos y drivers son más comunes en Windows - pues un driver para Windows 2000 en general será distinto al driver para Windows XP o Vista.

Linux en cambio, es un sistema de código abierto, los drivers son fácilmente creados por usuarios expertos en programación y las distribuciones de Linux, en general, vienen listas para poder utilizarse de forma casi automática. Cualquier componente de las computadoras actuales, pueden ser utilizados sin la necesidad de instalar los respectivos drivers.

En definitiva los drivers contienen el código dependiente del dispositivo, cada driver maneja un tipo de dispositivo, traduciendo los requerimientos abstractos en los comandos para el dispositivo:

- Escribe sobre los registros del controlador.
- Accesos a la memoria mapeada.
- Encola requerimientos.

Comúnmente las interrupciones de los dispositivos están asociadas a una función del driver. Estos sirven como una interfaz entre el sistema operativo y el hardware sobre en el que corren.

Los drivers forman parte del espacio de memoria del *kernel*, en general se cargan como módulos. Los fabricantes del hardware implementan el driver en función de una API especificada por el sistema operativo, escribiendo cómo lleva a cabo el dispositivo las solicitudes de la CPU.

Para agregar un nuevo hardware sólo basta con indicar el driver correspondiente sin necesidad de cambios en el *kernel*.

Nota: ejemplo de driver en Linux en PDF I/O 2, dispositivas 21, 22, 23, 24 y 25.

Performance: la E /S es uno de los factores que más afectan al rendimiento del sistema. Estas operaciones imponen una intensa demanda a la CPU para ejecutar código de los controladores de dispositivo y para planificar los procesos de forma equitativa y eficiente a medida que se bloquean

y desbloquean. Los cambios de contexto resultantes imponen una gran carga de trabajo a la CPU y a sus cachés hardware. Las operaciones de E/S también ponen al descubierto cualquier falta de eficiencia que exista en los mecanismos de tratamiento de interrupciones del *kernel*. Además, la E/S carga al bus de memoria durante la copia de datos entre las controladoras y la memoria física y, de nuevo, durante las copias entre los búferes del *kernel* y el espacio de datos de las aplicaciones. Tratar de satisfacer adecuadamente todas esas demandas es una de las principales preocupaciones de los arquitectos de un sistema informático.

Aunque las computadoras modernas pueden gestionar muchos miles de interrupciones por segundo, el tratamiento de interrupciones es una tarea relativamente cara en términos de procesamiento: cada interrupción hace que el sistema realice un cambio de estado, ejecute la rutina de tratamiento de la interrupción y luego restaure el estado. Las operaciones de E/S programadas pueden ser más eficientes que las E/S dirigidas por interrupciones, si el número de ciclos invertidos en las esperas activas no resulta excesivo. La terminación de una operación de E/S hace, típicamente, que se desbloquee un proceso, lo que provoca la sobrecarga asociada al correspondiente cambio de contexto.

El tráfico de red también puede provocar una alta tasa de cambios de contexto. Considere, por ejemplo, un inicio de sesión remoto en una máquina desde otra. Cada carácter escrito en la máquina local debe transportarse hasta la máquina remota. En la máquina local, el carácter se escribe en el teclado, se genera una interrupción de teclado y el carácter se pasa a través de la rutina de tratamiento de interrupción al controlador de dispositivo, al *kernel* y luego al proceso de usuario. El proceso de usuario ejecuta una llamada de E/S de red al sistema para enviar el carácter a la página remota. El carácter fluye entonces hacia el *kernel* local, a través de los niveles de red que construyen un paquete de red y hacia el controlador del dispositivo de red. El controlador del dispositivo de red transfiere el paquete a la tarjeta controladora de red, que envía el carácter y genera una interrupción. La interrupción pasa de nuevo a través del *kernel* para hacer que se complete la llamada de E/S de red al sistema.

Ahora, el hardware de red del sistema remoto recibe el paquete y se genera una interrupción. El carácter se desempaquetá segú los protocolos de red y se entrega al demonio de red apropiado. El demonio de red identifica qué sesión remota es la implicada y pasa el paquete al sub-demonio apropiado para dicha sesión. A lo largo de este flujo, se producen cambios de contexto y cambios de estado. Usualmente, el receptor devuelve el carácter como eco al transmisor, y dicha operación duplica la cantidad de trabajo que hay que realizar.

Para eliminar los cambios de contextos necesarios para mover cada carácter entre los demonios del *kernel*, los desarrolladores de Solaris reimplementaron el demonio telnet utilizando hebras internas al *kernel*. Sun estima que esta mejora permite incrementar el número máximo de inicios de sesión de red desde unos cuantos centenares a unos cuantos miles en un servidor de gran tamaño.

Otros sistemas utilizan procesadores frontales separados para la E/S de terminales con el fin de reducir la carga de interrupciones en la CPU principal. Por ejemplo, un concentrador de terminales puede multiplexar el tráfico de centenares de terminales remotos con un único puerto en una computadora de gran tamaño. Un canal de E/S es una CPU dedicada, de propósito especial, que podemos encontrar en las computadoras de tipo mainframe y en otros sistemas de alta gama. La tarea de un canal es descargar el trabajo de E/S de la CPU principal. La idea es que los canales hacen que los datos fluyan de manera suave, mientras que la CPU principal queda libre para procesar los datos. Al igual que las controladoras de dispositivos y las controladoras de DMA que podemos encontrar en computadoras de menor tamaño, un canal puede procesar programas más generales y sofisticados, de modo que los canales pueden optimizarse para ciertos tipos de cargas de trabajo. Podemos emplear diversos principios para mejorar la eficiencia de la E/S:

- Reducir el número de cambios de contexto, context switches.
- Reducir la cantidad de copias de los datos mientras se pasan del dispositivo a la aplicación.
- Reducir la frecuencia de las interrupciones utilizando:
 - Transferencias de gran cantidad de datos.
 - Controladoras más inteligentes.
 - Polling, si se minimiza la espera activa.
- Utilizar DMA.

Archivos: las computadoras pueden almacenar información en varios soportes de almacenamiento, como discos magnéticos, cintas magnéticas y discos ópticos. Para que el sistema informático sea cómodo de utilizar, el sistema operativo proporciona una vista lógica uniforme para el almacenamiento de la información. El sistema operativo realiza una abstracción de las propiedades físicas de los dispositivos de almacenamiento, con el fin de definir una unidad lógica de almacenamiento, el archivo. Los archivos son mapeados por el sistema operativo sobre los dispositivos físicos. Estos dispositivos de almacenamiento son, usualmente, no volátiles; de modo que los contenidos persisten aunque se produzcan fallos de alimentación o reinicios del sistema. Un archivo es una colección de información relacionada, con un nombre, que se graba en almacenamiento secundario. Desde la perspectiva del usuario, un archivo es la unidad más pequeña de almacenamiento secundario lógico. Comúnmente, los archivos representan programas y datos.

Desde el punto de vista del usuario las operaciones que puede llevar a cabo son:

- Nombrar un archivo.
- Asegurar la protección.
- Compartir archivos.
- No tratar aspectos físicos.

Desde el punto de vista del diseño, se puede:

- Implementar archivos.
- Implementar directorios.
- Manejo del espacio en disco.
- Manejo del espacio libre.
- Eficiencia y mantenimiento.

Manejo de archivos: un archivo es un tipo abstracto de datos. Para definir un archivo apropiadamente, debemos considerar las operaciones que pueden realizarse con los archivos. El sistema operativo puede proporcionar llamadas al sistema para:

- **Creación de un archivo:** para crear un archivo hace falta ejecutar dos pasos. En primer lugar, es necesario encontrar espacio para el archivo dentro del sistema de archivos. En segundo lugar, es necesario incluir en el directorio una entrada para el nuevo archivo.
- **Escritura en un archivo:** para escribir en un archivo, debemos realizar una llamada al sistema que especifique tanto el nombre del archivo como la información que hay que escribir en el archivo. Dado el nombre del archivo, el sistema explora el directorio para hallar la ubicación del archivo. El sistema debe mantener un puntero de escritura que haga referencia a la ubicación dentro del archivo en la que debe tener lugar la siguiente escritura. El puntero de escritura debe actualizarse cada vez que se escriba en el archivo nueva información.
- **Lectura de un archivo:** para leer desde un archivo, utilizamos una llamada al sistema que especifica el nombre del archivo y dónde debe colocarse, dentro de la memoria, el siguiente bloque del archivo. De nuevo, exploramos el directorio para hallar la entrada

asociada y el sistema necesitará mantener un puntero de lectura que haga referencia a la ubicación dentro del archivo en la que tiene que tener lugar la siguiente lectura. Una vez que la lectura se ha completado, se actualiza el puntero de lectura. Puesto que los procesos, usualmente, están o bien leyendo del archivo o bien escribiendo en él, se puede almacenar la ubicación correspondiente a la operación actual en un puntero de posición actual del archivo que será diferente para cada proceso. Las operaciones de lectura y escritura utilizan el mismo puntero, ahorrando así espacio y reduciendo la complejidad del sistema.

- **Reposiciónamiento dentro de un archivo:** se explora el directorio para hallar la correspondiente entrada y se reposiciona el puntero de posición actual dentro de un archivo, asignándole un nuevo valor. El reposicionamiento dentro del archivo no tiene por qué implicar que se realice ninguna operación de E/S. Esta operación de archivo se conoce también con el nombre de búsqueda en el archivo.
- **Borrado de un archivo:** para borrar un archivo, exploramos el directorio en busca del archivo indicado. Habiendo hallado la entrada de directorio asociada, liberamos todo el espacio del archivo, de modo que pueda ser reutilizado por otros archivos, y borramos también la propia entrada del directorio.
- **Truncado de un archivo:** el usuario puede querer borrar el contenido de un archivo, pero manteniendo sus atributos. En lugar de forzar al usuario a borrar el archivo y volverlo a crear, esta función permite que los atributos no se vean modificados, excepto la longitud del archivo, mientras que el archivo se reinicializa asignándole una longitud igual a cero y liberando el espacio que tuviera asignado.

Estas funciones facilitan el acceso a los archivos por parte de las aplicaciones, ofreciendo una abstracción al programador, en cuanto al acceso de bajo nivel. El programador no desarrolla el software de administración de archivos.

El sistema operativo debe:

- Cumplir con la gestión de datos.
- Cumplir con las solicitudes del usuario.
- Minimizar/eliminar la posibilidad de perder o destruir datos. Garantizando la integridad del contenido de los archivos.
- Dar soporte de E/S a distintos dispositivos.
- Brindar un conjunto de interfaces de E/S para tratamiento de archivos.
- Proveer soporte de E/S para múltiples usuarios.

Un archivo tiene una determinada estructura definida que dependerá de su tipo, existen dos tipos de archivos:

- I. **Archivos regulares:** son archivos que almacenan datos u objetos de un programa. Estos pueden ser:
 - **Texto plano:** un archivo de texto es una secuencia de caracteres organizada en líneas, y posiblemente en páginas. Source file.
 - **Binarios:** los cuales se dividen en:
 - **Executable file, archivo ejecutable:** un archivo fuente es una secuencia de subrutinas y funciones, cada una de las cuales está a su vez organizada como una serie de declaraciones, seguidas de instrucciones ejecutables.
 - **Object file, archivos de objetos:** un archivo objeto es una secuencia de bytes organizada en bloques que el programa montador del sistema puede comprender. Un archivo ejecutable es una serie de secciones de código que el cargador puede cargar en memoria y ejecutar.
- II. **Directarios:** son archivos que mantienen la estructura en el Fyle System.

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

- **Nombre:** el nombre de archivo simbólico es la única información que se mantiene en un formato legible por parte de las personas.
- **Identificador:** esta etiqueta unívoca, que usualmente es un número, identifica el archivo dentro del sistema de archivos; se trata de la versión no legible por las personas del nombre del archivo.
- **Tipo:** esta información es necesaria para los sistemas que soporten diferentes tipos de archivos.
- **Ubicación:** esta información es un puntero a un dispositivo y a la ubicación del archivo dentro de dicho dispositivo.
- **Tamaño:** este atributo expresa el tamaño actual del archivo y, posiblemente, el tamaño máximo permitido.
- **Protección:** información de control de acceso que determina quién puede leer el archivo, escribir en el archivo, ejecutarlo, etc.
 - Owner, permisos, password.
 - Momento en que el usuario lo modificó, creó, accedió por última vez.
 - ACLs.
- **Fecha, hora e identificación del usuario:** esta información puede mantenerse para los sucesos de creación, de última modificación y de último uso del archivo. Estos datos pueden resultar útiles para propósitos de protección, seguridad y monitorización del uso del archivo.

Directorios: asociado con cualquier sistema de gestión de archivos o cualquier colección de archivos suele haber un directorio de archivos. El directorio contiene información sobre los archivos, incluyendo atributos, ubicación y propietario. Gran parte de esta información, especialmente la relativa al almacenamiento, la gestiona el sistema operativo. El directorio es propiamente un archivo, poseído por el sistema operativo y accesible a través de diversas rutinas de gestión de archivos. Aunque parte de la información de los directorios está disponible para los usuarios y aplicaciones, en general, la información se proporciona indirectamente, a través de rutinas del sistema. De este modo, los usuarios no pueden acceder directamente al directorio, incluso en modo de sólo lectura. Desde el punto de vista del usuario, el directorio ofrece una traducción entre los nombres de archivo conocidos para usuarios y aplicaciones y los archivos, propiamente dicho. Por tanto, cada entrada incluirá el nombre del archivo. Casi todos los sistemas trabajan con clases diferentes de archivos y diferentes organizaciones de archivos, por lo que también se incluye esta información. Las acciones que se pueden llevar a cabo con un directorio son:

- Search for a file, buscar por archivo.
- Create a file (directory entry), crear un archivo.
- Delete a file (directory entry), borrar un archivo.
- List a directory, listar un directorio,
- Rename a file, renombrar archivos.

El uso de directorios ayuda con la eficiencia, ya que localiza más rápidamente a los archivos.

Permiten que diferentes usuarios tengan el mismo nombre de archivo, sin entrar en conflicto.

También agrupa lógicamente los archivos por propiedades/funciones, como por ejemplo programas java, juegos, bibliotecas, etc.

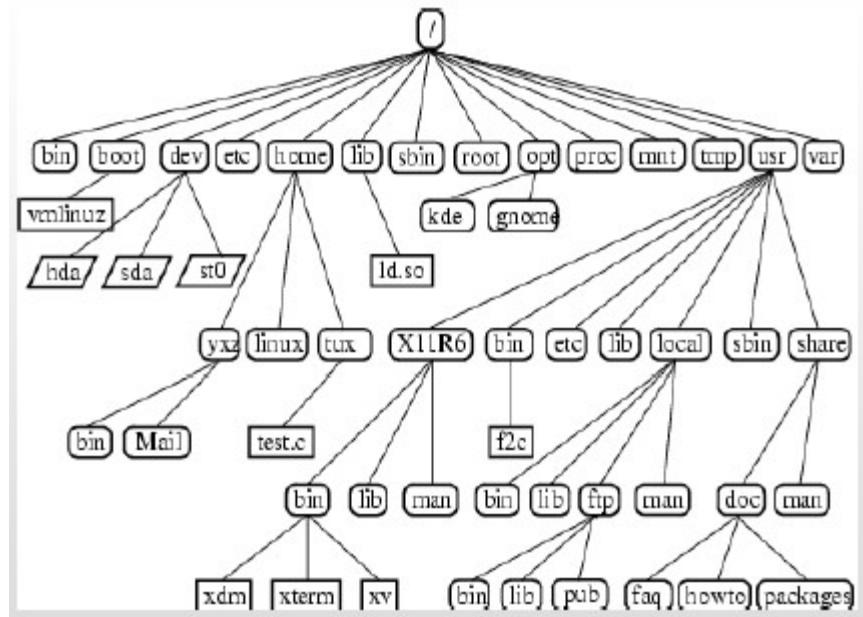
Estructura: la forma más simple de estructuración de un directorio es una lista de entradas, una para cada archivo. Esta estructura puede representarse con un simple archivo secuencial, con el nombre del archivo haciendo las veces de clave. En algunos sistemas antiguos monousuario se ha usado esta técnica. Sin embargo, no es adecuada cuando múltiples usuarios comparten el sistema e incluso para un solo usuario con muchos archivos.

Un método más potente y flexible, adoptado casi universalmente, es el directorio jerárquico o estructura en árbol. Existe un directorio maestro que contiene un número determinado de directorios de usuario. Cada uno de estos directorios puede tener a su vez subdirectorios y archivos como entradas. Esto se cumple en cualquier nivel. Es decir, en cualquier nivel, un directorio puede constar de entradas para subdirectorios y/o entradas para archivos.

Los archivos pueden ubicarse siguiendo un path desde el directorio raíz y sus sucesivas referencias, full pathname del archivo o path absoluto. Distintos archivos pueden tener el mismo nombre pero el fullpathname es único.

Aunque el nombre de camino facilita la elección de los nombres de archivo, para un usuario sería incómodo tener que deletrear el nombre de camino entero cada vez que haga una referencia a un archivo. Normalmente, cada usuario interactivo o proceso tiene asociado un directorio actual, conocido a menudo como directorio de trabajo. Las referencias a los archivos son entonces relativas al directorio de trabajo. Dentro del directorio de trabajo, se pueden referenciar los archivos tanto por:

- **Path absoluto:** el nombre incluye todo el camino del archivo. Por ejemplo /var/www/index.html, en Linux o C:\windows\winhelp.exe en Windows.
- **Path relativo:** el nombre se calcula relativamente al directorio en el que se encuentra. Si en el directorio /home/juan, se puede buscar ../../var/www/index.html.



Compartir archivos: en un sistema multiusuario casi siempre existe la necesidad de permitir a los usuarios compartir archivos. Se deben ofrecer la protección al propietario/administrador de controlar qué se puede hacer, derechos de acceso, y quién lo puede hacer. Emergen entonces dos cuestiones: los derechos de acceso y la gestión de los accesos simultáneos.

- I. **Derechos de acceso:** el sistema de archivos debe ofrecer una herramienta flexible para permitir la compartición general de archivos entre los usuarios, así como un conjunto de opciones de forma que se pueda controlar la manera en que se accede a cada archivo en particular. Normalmente, a los usuarios o grupos de usuarios le son concedidos ciertos derechos de acceso a cada archivo. Se ha venido usando un amplio rango de derechos de acceso. La lista siguiente es representativa de los derechos de acceso que pueden asignarse a un usuario particular para un archivo específico.

- **Ninguno:** el usuario no puede siquiera conocer la existencia del archivo.

- **Conocimiento:** el usuario puede determinar que el archivo existe y quién es su propietario.
- **Ejecución:** el usuario puede arrancar y ejecutar un programa pero no puede copiarlo.
- **Lectura:** el usuario puede leer el archivo para cualquier propósito.
- **Adición:** el usuario puede añadir datos al archivo, generalmente al final.
- **Actualización:** el usuario puede modificar, borrar y añadir datos al archivo.
- **Cambio de protección:** el usuario puede cambiar los derechos de acceso otorgados a otros usuarios.
- **Borrado:** el usuario puede borrar el archivo del sistema de archivos.

El propietario, owner, es un principio que posee todos los permisos. Este puede otorgar derechos a los otros usuarios. Puede ofrecerse acceso a las siguientes clases de usuarios:

- **Usuario específico:** usuarios individuales designados por su ID de usuario.
- **Grupos de usuarios:** un conjunto de usuarios no definidos individualmente. El sistema deberá disponer de algún medio para guardar constancia de la militancia de los grupos de usuarios.
- **Todos:** todos los usuarios que tengan acceso al sistema. Estos serán archivos públicos.

- II. **Accesos simultáneos:** cuando se otorga acceso para añadir o actualizar un archivo a más de un usuario, el sistema operativo o el sistema de gestión de archivos debe hacer cumplir una disciplina. Un método de fuerza bruta consiste en permitir a los usuarios bloquear el archivo entero cuando lo vaya a actualizar. Un mejor control es bloquear los registros individuales durante la actualización.

Almacenamiento de archivos: las metas de sistema e archivos son dos brindar espacio en disco a los archivos de usuario y del sistema, y mantener un registro del espacio libre. Cantidad y su ubicación dentro del mismo. Para llevar a cabo el almacenamiento se deben tener en cuenta diferentes conceptos:

- **Sector:** unidad de almacenamiento utilizada en los discos rígidos.
- **Bloque/Cluster:** conjunto de sectores consecutivos.
- **File System:** define la forma en que los datos son almacenados.
- **FAT, File Allocation Table:** contiene información sobre en qué lugar están alojados los distintos archivos.

Existen dos técnicas para llevar a cabo el almacenamiento, una asignación previa o preasignación y una asignación dinámica. Una política de asignación previa requeriría que el tamaño máximo de un archivo se declarase en el momento de crearlo. En algunos casos, como al compilar los programas, al crear archivos de datos de resumen o al transferir un archivo desde otro sistema por una red de comunicaciones, este valor puede estimarse. Sin embargo, para muchas aplicaciones es difícil, si no imposible, estimar de manera fiable el posible tamaño máximo del archivo. En esos casos, los usuarios y programadores de aplicaciones se inclinarían por sobreestimar el tamaño del archivo de forma que no se quedaran sin espacio. Evidentemente, esto es un derroche desde el punto de vista de la asignación de memoria secundaria, este método requiere que los sectores se alojen de manera contigua. Por tanto, existen ventajas en el uso de la asignación dinámica, que asigna espacio a los archivos en secciones a medida que se necesitan, en esta técnica no es necesario que los bloques se almacenen de manera contigua.

Métodos de asignación: después de una discusión previa con las formas de asignar, se pueden considerar métodos específicos de asignación de archivos:

- I. **Asignación contigua o continua:** cuando se crea un archivo se le asigna un único conjunto contiguo de bloques. Por tanto, ésta es una estrategia de asignación previa que emplea secciones de tamaño variable. La tabla de asignación de archivos necesita sólo una entrada por cada archivo, que muestre el bloque de comienzo y la longitud del archivo. La asignación contigua es la mejor desde el punto de vista de un archivo secuencial individual. Se pueden traer múltiples bloques de una vez para mejorar el rendimiento en los tratamientos secuenciales. También es fáciles recuperar un solo bloque. Este tipo de asignación presenta algunos problemas, ya que se producirá fragmentación externa, haciendo difícil encontrar bloques contiguos de espacio de tamaño suficiente. De cuando en cuando será necesario ejecutar un algoritmo de compactación para liberar espacio adicional en disco. Además para llevar a cabo esta técnica se requiere una pre-asignación por lo cual se debe conocer el tamaño previo del archivo. Su FAT es simple, sólo tiene una entrada que incluye un bloque de inicio y longitud.
- II. **Asignación encadenada:** normalmente, la asignación se hace con bloques individuales. Cada bloque contendrá un puntero al siguiente bloque de la cadena. La tabla de asignación de archivos, FAT, necesita de nuevo una sola entrada por cada archivo que muestre el bloque de comienzo y longitud de archivo. Aunque es posible la asignación previa, es más común simplemente asignar bloques a medida que se necesiten. La elección de los bloques es entonces una cuestión simple: cualquier bloque libre puede añadirse a la cadena. No hay que preocuparse por la fragmentación externa porque sólo se necesita un bloque cada vez. Este tipo de organización física se ajusta mejor a los archivos secuenciales que van a ser procesados secuencialmente, no random. Para seleccionar un bloque individual de un archivo se requiere recorrer la cadena hasta el bloque deseado, ya que no necesariamente tienen que estar contiguos. Aunque esto puede lograrse fácilmente encadenando los bloques con su vecino.
- III. **Asignación indexada:** trata muchos de los problemas de las asignaciones contigua y encadenada. En este caso, la tabla de asignación de archivos, FAT, contiene un índice separado de un nivel para cada archivo; el índice posee una entrada para cada sección asignada al archivo. Normalmente, los índices no están almacenados físicamente como parte de la tabla de asignación de archivos. Más exactamente, el índice del archivo se guardará en un bloque aparte y la entrada del archivo en la tabla de asignación apuntará a dicho bloque. La asignación puede hacerse por bloques de tamaño fijo o secciones de tamaño variables. La asignación por bloques elimina la fragmentación externa, mientras que la asignación por secciones mejora la cercanía. En cualquier caso, los archivos pueden concentrarse en zonas cercanas de cuando en cuando. La concentración reduce el tamaño del índice en el caso de secciones de tamaño variable, pero no en el caso de asignación por bloques. La asignación indexada soporta tanto el acceso secuencial como el acceso directo a los archivos y por ello se ha convertido en la forma más popular de asignación de archivos.

Gestión de espacio libre: se debe gestionar el espacio que no queda asignando actualmente a ningún archivo. Para llevar a cabo cualquiera de las técnicas descriptas, es necesario saber qué bloques del disco están disponibles. Por lo tanto, hace FAT de discos además de una FAT de archivos. Tres técnicas son de uso común:

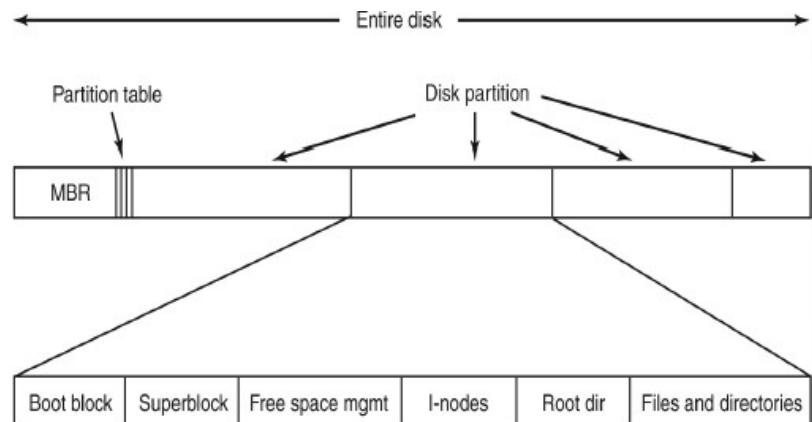
- I. **Tablas de bits:** un vector contiene un bit por cada bloque del disco. Cada entrada igual a 0 corresponde a un bloque libre y cada 1 corresponde a un bloque en uso. Las tablas de bits tienen la ventaja de que es relativamente fácil encontrar un bloque o un grupo contiguo de bloques libres. Están trabajan bien con cualquiera de los métodos de asignación de archivos

discutidos. Otra ventaja es que puede ser pequeña y puede mantenerse en memoria principal. Esto evita la necesidad de traer la FAT de disco a memoria cada vez que se realice una asignación.

- II. **Secciones libres encadenadas:** pueden encadenarse juntas mediante un puntero y un valor de longitud en cada sección libre. Este método tiene un gasto insignificante porque no hay necesidad de tabla de asignación de disco, sino simplemente un puntero al comienzo de la cadena y la longitud de la primera sección. Este método sirve para todas las técnicas de asignación de archivos. Si la asignación se realiza por bloques, solamente hay que elegir el bloque libre del principio de la cadena y retocar el primer puntero o el valor de longitud. Si la asignación se hace por secciones de longitud variable, puede usarse el algoritmo del primer hueco. Se traen la cabeceras de la sección una a una para determinar la siguiente sección de la cadena de ajuste. De nuevo, hay que retocar el puntero y las longitudes.
- III. **Indexación:** trata el espacio libre como si fuera un archivo y utiliza una tabla índice. Por rayones de eficiencia, el índice debe trabajar con secciones de tamaño variable mejor que con bloques. De este modo, habrá una entrada en la tabla para cada sección libre del disco. Este procedimiento ofrece un soporte eficaz para todos los métodos de asignación de archivos. Es una variante de bloques libres encadenados, en ella el primer bloque libre contiene las direcciones de N bloques libres. Las N-1 primeras direcciones son bloques libres, la N-ésima dirección referencia otro bloque con N direcciones de bloques libres. Existe una variante de indexación que es el recuento. Esta estrategia considera las situaciones de que varios bloques contiguos pueden ser solicitados o libreados a la vez, en especial con asignación contigua. En lugar de tener N direcciones libres, índice, se tiene:
 - La dirección del primer bloque libre.
 - Los N bloques contiguos que le siguen, (#bloque, N siguientes bloques libres)

Manejo de archivo Unix: el kernel de Unix contempla todos los archivos como flujos de bytes. Cualquier estructura lógica interna será específica de la aplicación. Sin embargo, Unix se ocupa de la estructura física de los archivos y distingue los siguientes tipos de archivos:

- Archivo común.
- Directorio.
- Archivos especiales, dispositivos /dev/sda.
- Named pipes, comunicación entre procesos.
- Links, comparten el i-nodo, solo dentro del mismo filesystem.
- Links simbólicos, para filesystems diferentes.



Todos los tipos de archivos de UNIX son administrados por el sistema operativo por medio de nodos-i. Un nodo-i, nodo índice, es una estructura de control que contiene la información clave de un archivo necesaria para el sistema operativo. Pueden asociarse varios nombres de archivo a un mismo nodo-i. Pero un nodo-i activo se puede asociar con un único archivo y cada archivo es controlado por un solo nodo-i. Los atributos del archivo, así como sus permisos y otra información de control, se almacenan en el nodo-i, se muestra en la siguiente imagen.

Modo de Archivo	Indicador (<i>flag</i>) de 16 bits que guarda los permisos de acceso y ejecución asociados con el archivo.
	12-14 Tipo de archivo (regular, directorio, especial de caracteres o de bloques, tubo FIFO)
	9-11 Indicadores de ejecución
	8 Permiso de Lectura para el Propietario
	7 Permiso de Escritura para el Propietario
	6 Permiso de Ejecución para el Propietario
	5 Permiso de Lectura para el Grupo
	4 Permiso de Escritura para el Grupo
	3 Permiso de Ejecución para el Grupo
	2 Permiso de Lectura para el Resto
	1 Permiso de Escritura para el Resto
	0 Permiso de Ejecución para el Resto
Cuenta de Enlaces	Número de referencias al nodo-i en los directorios
ID del Propietario	Propietario individual del archivo
ID del Grupo	Grupo propietario asociado al archivo
Tamaño de Archivo	Número de bytes del archivo
Direcciones del Archivo	39 bytes de información de direccionamiento
Último Acceso	Fecha del último acceso al archivo
Última Modificación	Fecha de la última modificación del archivo
<u>Modificación del Nodo-i</u>	<u>Fecha de la última modificación del nodo-i</u>

El número total de bloques de datos de un archivo dependerá de la capacidad de los bloques de tamaño fijo del sistema. Este esquema tiene varias ventajas:

- Los nodos-i son de tamaño fijo y relativamente pequeños, por lo que pueden guardarse en memoria principal durante períodos largos.
- Se puede acceder a los archivos más pequeños con poca indexación o ninguna, reduciendo así el procesamiento y el tiempo de acceso al disco.
- El tamaño máximo teórico de un archivo es suficientemente grande como para satisfacer a casi todas las aplicaciones.

Manejo de archivos Linux: un sistema de archivos virtual, VFS, es una capa de abstracción encima de un sistema de archivos más concreto. El propósito de un VFS es permitir que las aplicaciones cliente tengan acceso a diversos tipos de sistemas de archivos concretos de una manera uniforme. Puede ser utilizada para tender un puente sobre las diferencias en los sistemas de archivos de Windows, de Mac OS y Unix, de modo que las aplicaciones pudieran tener acceso a archivos en los sistemas de archivos locales de esos tipos sin tener que saber a qué tipo de sistema de archivos están teniendo acceso.

Un VFS especifica una interfaz o un contrato entre el kernel y un sistema de archivos en concreto. Por lo tanto, es fácil agregar nuevos sistemas de archivos al kernel simplemente satisfaciendo el contrato. Los términos del contrato pueden volverse incompatibles de una versión a otra, lo que requeriría que sistemas de archivos concretos fuesen recompilados, y posiblemente modificados antes de la re-compilación, para permitirles trabajar con un nuevo lanzamiento del sistema operativo; o el proveedor del sistema operativo pueda realizar solamente cambios retro-compatibles al contrato, de modo que un sistema de archivos concreto construido para un lanzamiento dado del sistema operativo trabaje con las versiones futuras del mismo sistema operativo. Los objetos que componen a la VFS son:

- **Superblock object:** representa el file System montado.
- **Inode object:** representa un archivo.
- **D-entry object:** representa una entrada en un directorio.

- **File object:** representa un archivo abierto asociado a un proceso.

Manejo de archivos de Windows: los file System que soportan Windows son:

- **CD-ROM file System, CDFS:** es simplemente el formato de sistema de archivos usual para un CD de datos. Para leerlo es tan simple como meterlo a la unidad y acceder al disco.
- **Universal disk format, UDS:** es un sistema de archivos utilizado por varios sistemas operativos UNIX y POSIX. Es un derivado del Berkeley Fast File System, FFS, el cual es desarrollado desde FS UNIX.
- **File allocation table, FAT:** es un sistema de archivos utilizado originalmente por DOS y Windows 9x. Windows aún sigue soportando FAT file System debido a:
 - Compatibilidad con otro SO en sistemas multiboot.
 - Permite upgrades, mejoras, desde versiones anteriores.
 - El formato de dispositivos como diskettes.

Estos utilizan un mapa de bloques del sistema de archivos, llamadas FAT. La FAT tiene tantas entradas como bloques. Su duplicado y el directorio raíz se almacenan en los primeros sectores de la partición, su formato es:

Boot sector	File allocation table 1	File allocation table 2 (duplicate)	Root directory	Other directories and all files
-------------	-------------------------	-------------------------------------	----------------	---------------------------------

Se utiliza un esquema de asignación encadenada, con la única diferencia es que el puntero al próximo bloque está en el FAT y no en los bloques. Los bloques libres y dañados presentan un código especial.

Las distintas versiones FAT se diferencian por un número que indica la cantidad de bits que se usan para identificar diferentes bloques o clusters:

- **FAT-12:** las direcciones de bloque solamente contienen 12 bits, su límite es de 2^{12} clusters. Esto complica la implementación. El tamaño del disco se almacena como una cuenta de 16 bits expresada en sectores, lo que limita el espacio manejable a 32 megabytes. El habitual disquete constaba de 40 pistas con 8 sectores por pista, resultando en una capacidad inferior a 160 kilobytes. Este límite excedía la capacidad en más de un orden de magnitud, y al mismo tiempo, permitía encajar todas las estructuras de control en la primera pista. Por tanto, se evitaba el movimiento de los cabezales en las operaciones de lectura y escritura. Estos límites fueron superados en los años posteriores.
- **FAT-16:** se eliminó el contador de sectores de 16 bits. El tamaño de la partición ahora estaba limitado por la cuenta de sectores por clúster, que era de 8 bits. Esto obligaba a usar clusters de 32 KiB con los usuales 512 bytes por sector. Así que el límite definitivo de FAT16 se situó en los 90 GiB. Esta mejora estuvo disponible en 1988. Mucho más tarde, Windows XP aumentó el tamaño máximo del cluster a 64 kilobytes gracias al "truco" de considerar la cuenta de clusters como un entero sin signo. No obstante, el formato resultante no era compatible con otras implementaciones de la época, y además, generaba más fragmentación interna. Windows 98 fue compatible con esta extensión en lo referente a lectura y escritura. Sin embargo, sus utilidades de disco no eran capaces de trabajar con ella.
- **FAT-32:** fue la respuesta para superar el límite de tamaño de FAT16 al mismo tiempo que se mantenía la compatibilidad con MS-DOS en modo real. Microsoft decidió implementar una nueva generación de FAT utilizando direcciones de cluster de 32 bits, aunque sólo 28 de esos bits se utilizaban realmente. En teoría, esto debería permitir aproximadamente 100.100.538.948.585.453 clusters,

arrojando tamaños de almacenamiento cercanos a los 8 TiB. Sin embargo, debido a limitaciones en la utilidad ScanDisk de Microsoft, no se permite que FAT32 crezca más allá de 4.177.920 clusters por partición, es decir, unos 124 GiB.
FAT32 apareció por primera vez en Windows 95 OSR2.

Era necesario reformatear para usar las ventajas de FAT32. Windows 98 incorporó una herramienta para convertir de FAT16 a FAT32 sin pérdida de los datos. Este soporte no estuvo disponible en la línea empresarial hasta Windows 2000.

Block size	FAT-12	FAT-16	FAT-32
0.5 KB	2 MB		
1 KB	4 MB		
2 KB	8 MB	128 MB	
4 KB	16 MB	256 MB	1 TB
8 KB		512 MB	2 TB
16 KB		1024 MB	2 TB
32 KB		2048 MB	2 TB

- **New technology file System, NTFS:** es el file System nativo de Windows. Está basado en el sistema de archivos HPFS de IBM/Microsoft usado en el sistema operativo OS/2, y también tiene ciertas influencias del formato de archivos HFS diseñado por Apple. NTFS permite definir el tamaño del clúster a partir de 512 bytes, tamaño mínimo de un sector, de forma independiente al tamaño de la partición. Es un sistema adecuado para las particiones de gran tamaño requeridas en estaciones de trabajo de alto rendimiento y servidores. Puede manejar volúmenes de, teóricamente, hasta $2^{64}-1$ clústeres. En la práctica, el máximo volumen NTFS soportado es de $2^{32}-1$ clústeres, aproximadamente 16 TiB usando clústeres de 4 KiB.

Su principal inconveniente es que necesita para sí mismo una buena cantidad de espacio en disco duro, por lo que no es recomendable su uso en discos con menos de 400 MiB libres.

FAT es simple y más rápido para ciertas operaciones, pero NTFS soporta:

- Tamaños de archivo y discos mayores.
- Mejor performance en discos grandes.
- Nombres de archivos de hasta 255 caracteres.
- Atributos de seguridad.

Buffer cache: buffers en memoria principal para almacenamiento temporario de sectores de disco. Contienen una copia de algunos sectores de disco, su objetivo principal es minimizar la frecuencia de acceso al disco. Cuando un proceso quiere acceder a un bloque de la cache hay dos alternativas:

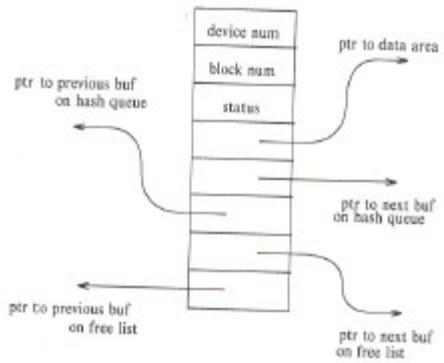
- I. Se copia al espacio de direcciones de usuario.
- II. Se trabaja como memoria compartida, no se copia permitiendo acceso a varios procesos.

Cuando se necesita un buffer para cargar un nuevo bloque, se elige el que hace más tiempo que no es referenciado. Es una lista de bloques, donde el último es el más recientemente usado, LRU. Cuando un bloque se referencia o entra en la cache queda al final de la lista, no se mueven los bloques en la memoria solamente se asocian punteros. Otra alternativa es la LFU, la cual se basa en reemplazar el que tenga menor número de referencias.

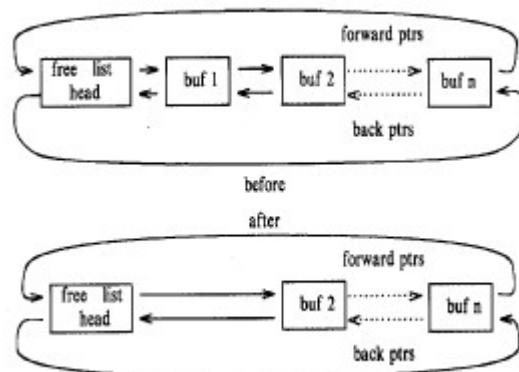
En Unix System v el buffer caché mantiene el objetivo de minimizar la frecuencia de acceso a disco. Es una estructura formada por buffers. El kernel asigna un espacio en la memoria durante la inicialización para esta estructura, un buffer tiene dos partes:

- I. **Header:** identifica por número de dispositivo y número de bloque, está compuesto de 5 punteros, 2 para la hash queue, 2 para la free list y un puntero al bloque en memoria y del estado actual. Donde los estados pueden ser:

- Free o disponible.
- Busy o no disponible, en uso por algún proceso.
- El kernel está escribiendo a disco o leyendo del disco.
- Delayed write: buffers que hayan sido modificados en memoria, pero el bloque original en disco todavía no fue actualizado.

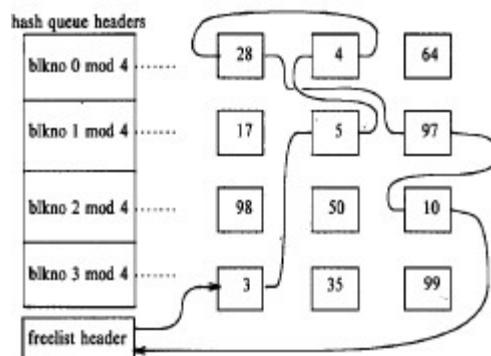


- II. **Free list:** organiza los buffers disponibles, es decir, los buffers donde se puede cargar un nuevo bloque de disco. No necesariamente los bloques están vacíos, esta lista se ordena según LRU.

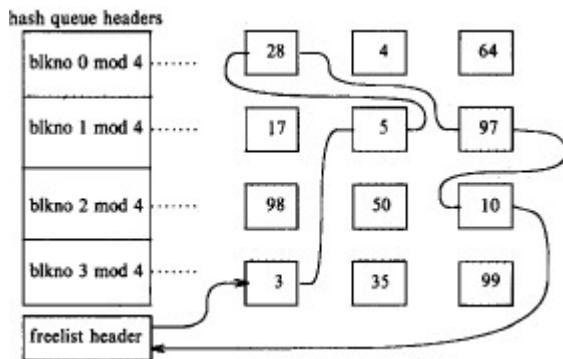


Búsqueda o recuperación de un buffer: como ya se dijo se utilizan hash queues, las cuales son colas para optimizar la búsqueda de un buffer en particular. Se organizan según una función de hash usando (dispositivo, #bloque). Para realizar la recuperación de un bloque se pueden presentar diferentes escenarios:

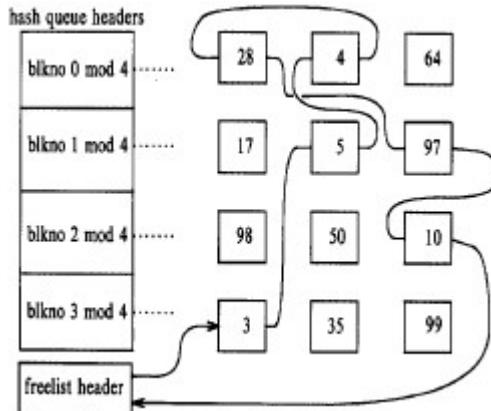
- I. El kernel encuentra el bloque en la hash queue, está disponible en la free list. Por ejemplo se busca el bloque 4.



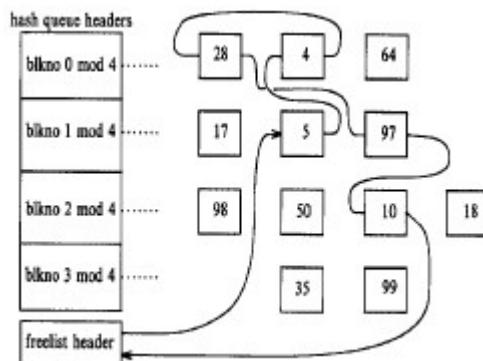
Se remueve ese buffer de la free list, pasando al estado busy y desde ahí el proceso puede usar el bloque.



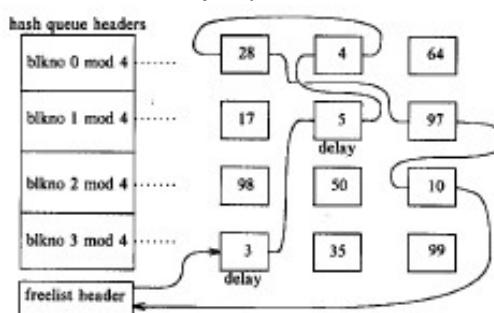
- II. El bloque buscado no está en la hash queue. Por ejemplo se busca el bloque 18.



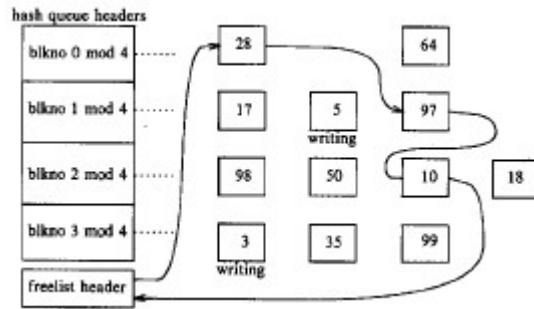
En este caso se toma un buffer de la free list, por ejemplo el 3 por lo general se usa el primero, se lee del disco el bloque deseado en el buffer obtenido y se ubica el hash queue correspondiente.



- III. El kernel no encuentra el bloque buscado en la hash queue. Se debe tomar el primero de la free list, pero está marcado DW. Por ejemplo se busca el 18 , toma el 3.



El kernel debe mandar ese bloque a disco y tomar otro buffer de la free list. Si también está DW, sigue hasta encontrar uno que no lo este. Mientras manda a escribir a disco los DW, asigna el siguiente free. Una vez escritos a disco los bloques DW, estos son ubicados al principio de la free list.



- IV. El kernel no encuentra el bloque en la hash queue y la free list está vacía. El proceso espera que se libere algún buffer. Cuando el proceso despierta se debe verificar nuevamente que el bloque no esté en la hash queue, algún proceso pudo haberlo cargado mientras éste dormía.
- V. El kernel busca un bloque y el buffer que lo contiene está marcado como busy. El proceso queda en espera.