Sistemas Operativos 2019

Resumen Final - Juan Ignacio Camou

Bibliografía

- Sistemas Operativos - William Stallings (5ta)
- Fundamentos de Sistemas Operativos - Abraham Silberschatz, Peter Baer Galvin & Greg Gagne (7ma).
- Sistemas Operativos, Diseño e Implementación - Andrew S. Tanenbaum y Albert S. Woodhul (2da)

Indice

1. Introducción a los Computadores

Un sistema operativo explota los recursos hardware de uno o más procesadores para proporcionar un conjunto de servicios a los usuarios del sistema. El sistema operativo también gestiona la memoria secundaria y los dispositivos de E/S (entrada/salida) para sus usuarios. Por tanto, es importante tener algunos conocimientos del hardware del computador subyacente antes de iniciar el estudio de los sistemas operativos.

1.1. Elementos Básicos

- **Procesador**. Controla el funcionamiento del computador y realiza sus funciones de procesamiento de datos. Cuando sólo hay un procesador, se denomina usualmente unidad central de proceso (Central Processing Unit, CPU).
- **Memoria principal**. Almacena datos y programas. Esta memoria es habitualmente volátil; es decir, cuando se apaga el computador, se pierde su contenido. En contraste, el contenido de la memoria del disco se mantiene incluso cuando se apaga el computador. A la memoria principal se le denomina también memoria real o memoria primaria.
- **Módulos de E/S.** Transfieren los datos entre el computador y su entorno externo. El entorno externo está formado por diversos dispositivos, incluyendo dispositivos de memoria secundaria (por ejemplo, discos), equipos de comunicaciones y terminales.
- **Bus del sistema**. Proporciona comunicación entre los procesadores, la memoria principal y los módulos de E/S.

1.2. Registros del Procesador

Un procesador incluye un conjunto de registros que proporcionan un tipo de memoria que es más rápida y de menor capacidad que la memoria principal. Los registros del procesador sirven para dos funciones:

Registros visibles para el usuario

Permiten al programador en lenguaje máquina o en ensamblador minimizar las referencias a memoria principal optimizando el uso de registros. Los tipos de registros que están normalmente disponibles son registros de datos y de dirección.

Los registros de datos para diversas funciones. En algunos casos, son de propósito general y pueden usarse con cualquier instrucción de máquina que realice operaciones sobre datos. Sin embargo, frecuentemente, hay restricciones. Por ejemplo, puede haber registros dedicados a operaciones de coma flotante y otros a operaciones con enteros.

Los registros de dirección contienen direcciones de memoria principal de datos e instrucciones, o una parte de la dirección que se utiliza en el cálculo de la dirección efectiva o completa. Algunos de ellos son:

- Registro índice. El direccionamiento indexado es un modo común de direccionamiento que implica sumar un índice a un valor de base para obtener una dirección efectiva.
- Puntero de segmento. Con direccionamiento segmentado, la memoria se divide en segmentos, que son bloques de palabras de longitud variable. Una referencia de memoria consta de una referencia a un determinado segmento y un desplazamiento dentro del segmento.
- Puntero de pila. Si hay direccionamiento de pila visible para el usuario, hay un registro dedicado que apunta a la cima de la pila.

Registro de Control y Estado

Usados por el procesador para controlar su operación y por rutinas privilegiadas del sistema operativo para controlar la ejecución de programas. Algunos de ellos son:

- Contador de programa (Program Counter, PC). Contiene la dirección de la próxima instrucción que se leerá de la memoria.
- Registro de instrucción (Instruction Register, IR). Contiene la última instrucción leída.

Todos los diseños de procesador incluyen también un registro, o conjunto de registros, conocido usualmente como la palabra de estado del programa (Program Status Word, PSW), que contiene información de estado. La PSW contiene normalmente códigos de condición, además de otra información de estado, tales como un bit para habilitar/inhabilitar las interrupciones y un bit de modo usuario/ supervisor.
Los códigos de condición (también llamados indicadores) son bits cuyo valor lo asigna normalmente el hardware de procesador teniendo en cuenta el resultado de las operaciones. Los bits de código de condición se agrupan en uno o más registros. Normalmente, forman parte de un registro de control.

En máquinas que utilizan múltiples tipos de interrupciones, se puede proporcionar un conjunto de registros de interrupciones, con un puntero a cada rutina de tratamiento de interrupción. Si se utiliza una pila para implementar ciertas funciones (por ejemplo llamadas a procedimientos), se necesita un puntero de pila de sistema. El hardware de gestión de memoria, requiere registros dedicados. Asimismo, se pueden utilizar registros en el control de las operaciones de E/S.

Además podemos nombrar los siguientes registros: un registro de dirección de memoria (RDIM), que especifica la dirección de memoria de la siguiente lectura o escritura; y un registro de datos de memoria (RDAM), que contiene los datos que se van a escribir en la memoria o que recibe los datos leídos de la memoria. De manera similar, un registro de dirección de E/S (RDIE/S) especifica un

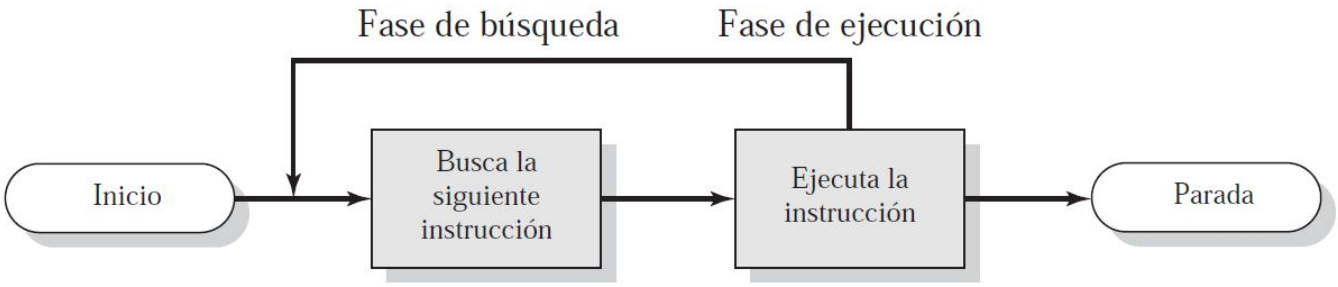
determinado dispositivo de E/S, y un registro de datos de E/S (RDAE/S) permite el intercambio de datos entre un módulo de E/S y el procesador.

1.3. Ejecución de Instrucciones

En su forma más simple, el procesamiento de una instrucción consta de dos pasos: el procesador lee (busca) instrucciones de la memoria, una cada vez, y ejecuta cada una de ellas. La ejecución de la instrucción puede involucrar varias operaciones dependiendo de la naturaleza de la misma.

Se denomina ciclo de instrucción al procesamiento requerido por una única instrucción. Se describe el ciclo de instrucción utilizando la descripción simplificada de dos pasos. A estos dos pasos se les denomina fase de búsqueda y de ejecución. La ejecución del programa se detiene sólo si se apaga la máquina, se produce algún tipo de error irrecuperable o se ejecuta una instrucción del programa que para el procesador.

Búsqueda y Ejecución de una Instrucción



Al principio de cada ciclo de instrucción, el procesador lee una instrucción de la memoria. En un procesador típico, el contador del programa (PC) almacena la dirección de la siguiente instrucción que se va a leer. A menos que se le indique otra cosa, el procesador siempre incrementa el PC después de cada instrucción ejecutada, de manera que se leerá la siguiente instrucción en orden secuencial.

La instrucción leída se carga dentro de un registro del procesador conocido como registro de instrucción (IR). La instrucción contiene bits que especifican la acción que debe realizar el procesador. El procesador interpreta la instrucción y lleva a cabo la acción requerida. En general, estas acciones se dividen en cuatro categorías:

- Procesador-memoria. Se pueden transferir datos desde el procesador a la memoria o viceversa.
- Procesador-E/S. Se pueden enviar datos a un dispositivo periférico o recibirlos desde el mismo, transfiriéndolos entre el procesador y un módulo de E/S.

- Procesamiento de datos. El procesador puede realizar algunas operaciones aritméticas o lógicas sobre los datos.
- Control. Una instrucción puede especificar que se va a alterar la secuencia de ejecución. Por ejemplo, el procesador puede leer una instrucción de la posición 149, que especifica que la siguiente instrucción estará en la posición 182. El procesador almacenará en el contador del programa un valor de 182. Como consecuencia, en la siguiente fase de búsqueda, se leerá la instrucción de la posición 182 en vez de la 150.

Una ejecución de una instrucción puede involucrar una combinación de estas acciones.

Sistemas de E/S

Se pueden intercambiar datos directamente entre un módulo de E/S (por ejemplo, un controlador de disco) y el procesador. El procesador identifica un dispositivo específico que está controlado por un determinado módulo de E/S.
Es deseable permitir que los intercambios de E/S se produzcan directamente con la memoria para liberar al procesador de la tarea de E/S. En tales casos, el procesador concede a un módulo de E/S la autorización para leer o escribir de la memoria, de manera que la transferencia entre memoria y E/S puede llevarse a cabo sin implicar al procesador. Durante dicha transferencia, el módulo de E/S emite mandatos de lectura y escritura a la memoria, liberando al procesador de la responsabilidad del intercambio. Esta operación, conocida como acceso directo a memoria (Direct Memory Access, DMA)

1.4. Interrupciones

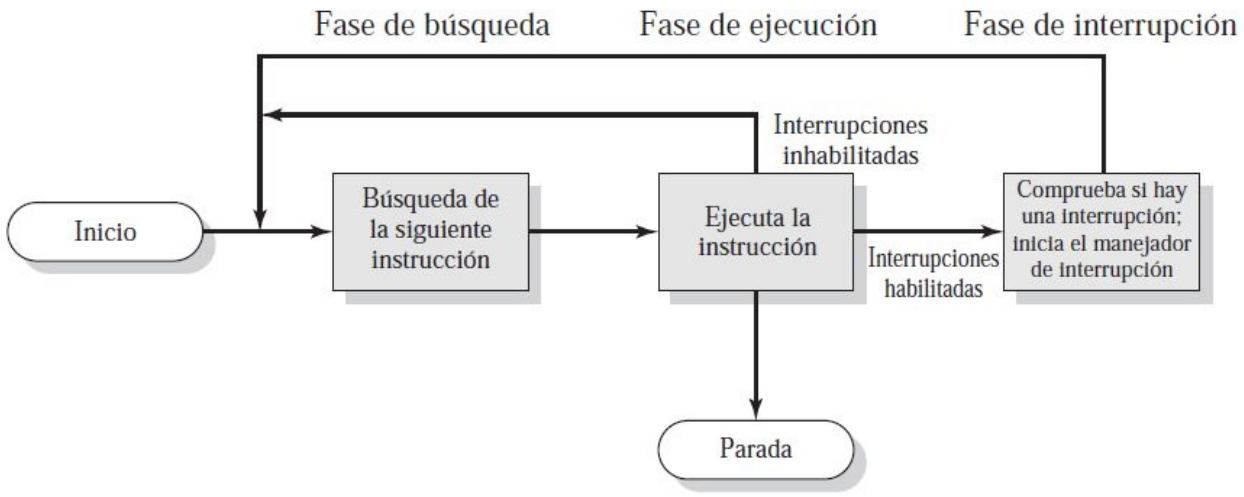
Prácticamente todos los computadores proporcionan un mecanismo por el cual otros módulos (memoria y E/S) pueden interrumpir el secuenciamiento normal del procesador. Básicamente, las interrupciones constituyen una manera de mejorar la utilización del procesador.

Tipos

- De programa. Generada por alguna condición que se produce como resultado de la ejecución de una instrucción, tales como un desbordamiento aritmético, una división por cero, un intento de ejecutar una instrucción de máquina ilegal, y las referencias fuera del espacio de la memoria permitido para un usuario.
- Por temporizador. Generada por un temporizador del procesador. Permite al sistema operativo realizar ciertas funciones de forma regular.
- De E/S. Generada por un controlador de E/S para señalar la conclusión normal de una operación o para indicar diversas condiciones de error.
- Por fallo del hardware. Generada por un fallo, como un fallo en el suministro de energía o un error de paridad en la memoria.

Interrupciones y Ciclos de Instrucciones

Gracias a las interrupciones, el procesador puede dedicarse a ejecutar otras instrucciones mientras que la operación de E/S se está llevando a cabo.
De cara al programa de usuario, una interrupción suspende la secuencia normal de ejecución. Cuando se completa el procesamiento de la interrupción, se reanuda la ejecución . Por tanto, el programa de usuario no tiene que contener ningún código especial para tratar las interrupciones; el procesador y el sistema operativo son responsables de suspender el programa de usuario y, posteriormente, reanudarlo en el mismo punto.
Para tratar las interrupciones, se añade una fase de interrupción al ciclo de instrucción. En la fase de interrupción, el procesador comprueba si se ha producido cualquier interrupción, hecho indicado por la presencia de una señal de interrupción. Si no hay interrupciones pendientes, el procesador continúa con la fase de búsqueda y lee la siguiente instrucción del programa actual. Si está pendiente una interrupción, el procesador suspende la ejecución del programa actual y ejecuta la rutina del manejador de interrupción. La rutina del manejador de interrupción es generalmente parte del sistema operativo.



Normalmente, esta rutina determina la naturaleza de la interrupción y realiza las acciones que se requieran. El manejador determina qué módulo de E/S generó la interrupción y puede dar paso a un programa que escriba más datos en ese módulo de E/S. Cuando se completa la rutina del manejador de interrupción, el procesador puede reanudar la ejecución del programa de usuario en el punto de la interrupción.
Es evidente que este proceso implica cierta sobrecarga. Deben ejecutarse instrucciones adicionales (en el manejador de interrupción) para determinar la naturaleza de la interrupción y decidir sobre la acción apropiada. Sin embargo, debido a la cantidad relativamente elevada de tiempo que se gastaría simplemente a la espera de una operación de E/S, el procesador se puede emplear mucho más eficientemente con el uso de interrupciones.

Procesamiento de Interrupciones

La aparición de una interrupción dispara varios eventos, tanto en el hardware del procesador como en el software. Cuando un dispositivo de E/S completa una operación de E/S, se produce la siguiente secuencia de eventos en el hardware:

1. El dispositivo genera una señal de interrupción hacia el procesador.
2. El procesador termina la ejecución de la instrucción actual antes de responder a la interrupción.
3. El procesador comprueba si hay una petición de interrupción pendiente, determina que hay una y manda una señal de reconocimiento al dispositivo que produjo la interrupción. Este reconocimiento permite que el dispositivo elimine su señal de interrupción.
4. En ese momento, el procesador necesita prepararse para transferir el control a la rutina de interrupción. Para comenzar, necesita salvar la información requerida para reanudar el programa actual en el momento de la interrupción. La información mínima requerida es la palabra de estado del programa (PSW) y la posición de la siguiente instrucción que se va a ejecutar, que está contenida en el contador de programa. Esta información se puede apilar en la pila de control de sistema.
5. A continuación, el procesador carga el contador del programa con la posición del punto de entrada de la rutina de manejo de interrupción que responderá a esta interrupción. Dependiendo de la arquitectura de computador y del diseño del sistema operativo, puede haber un único programa, uno por cada tipo de interrupción o uno por cada dispositivo y tipo de interrupción. Si hay más de una rutina de manejo de interrupción, el procesador debe determinar cuál invocar. Esta información puede estar incluida en la señal de interrupción original o el procesador puede tener que realizar una petición al dispositivo que generó la interrupción para obtener una respuesta que contiene la información requerida.

Una vez que se ha cargado el contador del programa, el procesador continúa con el siguiente ciclo de instrucción, que comienza con una lectura de instrucción. Dado que la lectura de la instrucción está determinada por el contenido del contador del programa, el resultado es que se transfiere el control al programa manejador de interrupción. La ejecución de este programa conlleva las siguientes operaciones:

6. En este momento, el contador del programa y la PSW vinculados con el programa interrumpido se han almacenado en la pila del sistema. Sin embargo, hay otra información que se considera parte del estado del programa en ejecución. En concreto, se necesita salvar el contenido de los registros del procesador, puesto que estos registros los podría utilizar el manejador de interrupciones. Por tanto, se deben salvar todos estos valores, así como cualquier otra información de estado. Generalmente, el manejador de interrupción comenzará salvando el contenido de todos los registros en la pila.
7. El manejador de interrupción puede en este momento comenzar a procesar la interrupción.

8. Cuando se completa el procesamiento de la interrupción, se recuperan los valores de los registros salvados en la pila y se restituyen en los registros.
9. La última acción consiste en restituir de la pila los valores de la PSW y del contador del programa. Como resultado, la siguiente instrucción que se va ejecutar corresponderá al programa previamente interrumpido.

Es importante salvar toda la información de estado del programa interrumpido para su posterior reanudación. Esto se debe a que la interrupción no es una rutina llamada desde el programa. En su lugar, la interrupción puede suceder en cualquier momento y, por tanto, en cualquier punto de la ejecución de un programa de usuario. Su aparición es imprevisible.

Multiples Interrupciones

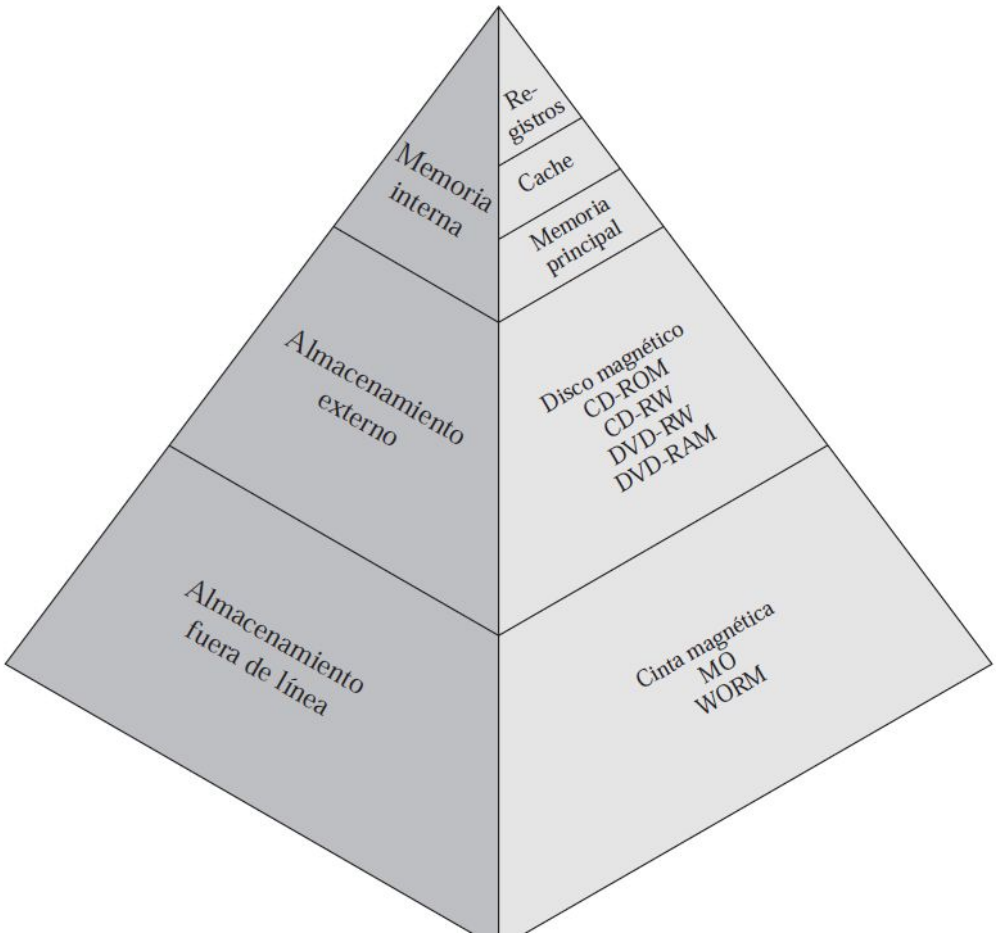
Se pueden considerar dos alternativas a la hora de tratar con múltiples interrupciones. La primera es inhabilitar las interrupciones mientras que se está procesando una interrupción. Una interrupción inhabilitada significa simplemente que el procesador ignorará cualquier nueva señal de petición de interrupción. Si se produce una interrupción durante este tiempo, generalmente permanecerá pendiente de ser procesada, de manera que el procesador sólo la comprobará después de que se rehabiliten las interrupciones. Por tanto, cuando se ejecuta un programa de usuario y se produce una interrupción, se inhabilitan las interrupciones inmediatamente.
Esta estrategia es válida y sencilla, puesto que las interrupciones se manejan en estricto orden secuencial. La desventaja es que no tiene en cuenta la prioridad relativa o el grado de urgencia de las interrupciones.
Una segunda estrategia es definir prioridades para las interrupciones y permitir que una interrupción de más prioridad cause que se interrumpa la ejecución de un manejador de una interrupción de menor prioridad.

Multiprogramación

Cuando el procesador trata con varios programas, la secuencia en la que se ejecutan los programas dependerá de su prioridad relativa, así como de si están esperando la finalización de una operación de E/S. Cuando se interrumpe un programa y se transfiere el control a un manejador de interrupción, una vez que se ha completado la rutina del manejador de interrupción, puede ocurrir que no se le devuelva inmediatamente el control al programa de usuario que estaba en ejecución en ese momento. En su lugar, el control puede pasar a algún otro programa pendiente de ejecutar que tenga una prioridad mayor. Posteriormente, se reanudará el programa de usuario interrumpido previamente, en el momento en que tenga la mayor prioridad.

1.5. Jerarquia de Memoria

Las restricciones de diseño en la memoria de un computador se pueden resumir en tres preguntas: ¿cuál es su capacidad? ¿Cuál es su velocidad? ¿Cuál es su coste? Las memorias más rápidas, caras y pequeñas se complementan con memorias más lentas, baratas y grandes.



Es posible organizar los datos a través de la jerarquía de manera que el porcentaje de accesos a cada nivel sucesivamente más bajo es considerablemente menor que al nivel inferior. El tipo de memoria más rápida, pequeña y costosa consiste en los registros internos del procesador.

Descendiendo dos niveles, está la memoria principal que es el sistema de memoria interna fundamental del computador. La memoria principal se amplía usualmente con una cache, más pequeña y de mayor velocidad. La cache normalmente no es visible al programador o, incluso, al procesador. Se trata de un dispositivo que controla el movimiento de datos entre la memoria principal y los registros de procesador con objeto de mejorar el rendimiento. Las tres formas de memoria descritas son, normalmente, volátiles y emplean tecnología de semi conductores.

Los datos se almacenan de forma más permanente en los dispositivos de almacenamiento masivo externos, de los cuales los más comunes son los discos duros y los dispositivos extraíbles, tales como los discos extraíbles, las cintas y el almacenamiento óptico. La memoria no volátil externa se denomina también memoria secundaria o memoria auxiliar.

1.6. Memoria Cache

La velocidad a la que el procesador puede ejecutar instrucciones está claramente limitada por el tiempo de ciclo de memoria (el tiempo que se tarda en leer o escribir una palabra de la memoria). Idealmente, se debería construir la memoria principal con la misma tecnología que la de los registros del procesador, consiguiendo tiempos de ciclo de memoria comparables a los tiempos de ciclo del procesador. Esta estrategia siempre ha resultado demasiado costosa. La solución consiste en aprovecharse del principio de la proximidad utilizando una memoria pequeña y rápida entre el procesador y la memoria principal, denominada cache.

El propósito de la memoria cache es proporcionar un tiempo de acceso a memoria próximo al de las memorias más rápidas disponibles y, al mismo tiempo, ofrecer un tamaño de memoria grande que tenga el precio de los tipos de memorias de semiconductores menos costosas.

Diseño de la Cache

Se deben tener en cuenta distintos conceptos al hablar de memoria cache. Se ha tratado ya el tema del tamaño de la cache, llegandose a la conclusión de que una cache de un tamaño razonablemente pequeño puede tener un impacto significativo en el rendimiento. Otro aspecto relacionado con la capacidad de la cache es el tamaño del bloque: la unidad de datos que se intercambia entre la cache y la memoria principal. Según el tamaño del bloque se incrementa desde muy pequeño a tamaños mayores, al principio la tasa de aciertos aumentará debido al principio de la proximidad: la alta probabilidad de que accedan en el futuro inmediato a los datos que están en la proximidad de una palabra a la que se ha hecho referencia. Según se incrementa el tamaño de bloque, se llevan a la cache más datos útiles.

Cuando se lee e incluye un nuevo bloque de datos en la cache, la función de correspondencia determina qué posición de la cache ocupará el bloque. Existen dos restricciones que afectan al diseño de la función de correspondencia. En primer lugar, cuando se introduce un bloque en la cache, se puede tener que reemplazar otro. Sería deseable hacer esto de manera que se minimizará la probabilidad de que se reemplázase un bloque que se necesitará en el futuro inmediato. Cuanto más flexible es la función de correspondencia, mayor grado de libertad a la hora de diseñar un algoritmo de reemplazo que maximice la tasa de aciertos. En segundo lugar, cuanto más flexible es la función de correspondencia, más compleja es la circuitería requerida para buscar en la cache y determinar si un bloque dado está allí.

El algoritmo de reemplazo selecciona, dentro de las restricciones de la función de correspondencia, qué bloque reemplazar cuando un nuevo bloque va a cargarse en la cache y ésta tiene todos los huecos llenos con otros bloques. Sería deseable reemplazar el bloque que menos probablemente se va a necesitar de nuevo en el futuro inmediato. Aunque es imposible identificar tal bloque, una

estrategia razonablemente eficiente es reemplazar el bloque que ha estado en la cache durante más tiempo sin haberse producido ninguna referencia a él. Esta política se denomina el algoritmo del menos recientemente usado (Least Recently Used, LRU). Se necesitan mecanismos hardware para identificar el bloque menos recientemente usado. Si se altera el contenido de un bloque en la cache, es necesario volverlo a escribir en la memoria principal antes de reemplazarlo.
La política de escritura dicta cuando tiene lugar la operación de escritura en memoria. Una alternativa es que la escritura se produzca cada vez que se actualiza el bloque. Otra opción es que la escritura se realice sólo cuando se reemplaza el bloque. La última estrategia minimiza las operaciones de escritura en memoria pero deja la memoria principal temporalmente en un estado obsoleto. Esto puede interferir con el modo de operación de un multiprocesador y con el acceso directo a memoria realizado por los módulos hardware de E/S.

1.7. Técnicas de Comunicación de E/S

Hay tres técnicas para llevar a cabo las operaciones de E/S:

E/S programada

Cuando el procesador ejecuta un programa y encuentra una instrucción relacionada con la E/S, ejecuta esa instrucción generando un mandato al módulo de E/S apropiado. En el caso de la E/S programada, el módulo de E/S realiza la acción solicitada y fija los bits correspondientes en el registro de estado de E/S, pero no realiza ninguna acción para avisar al procesador. En concreto, no interrumpe al procesador. Por tanto, después de que se invoca la instrucción de E/S, el procesador debe tomar un papel activo para determinar cuándo se completa la instrucción de E/S. Por este motivo, el procesador comprueba periódicamente el estado del módulo de E/S hasta que encuentra que se ha completado la operación.

Con esta técnica, el procesador es responsable de extraer los datos de la memoria principal en una operación de salida y de almacenarlos en ella en una operación de entrada. El sotfware de E/S se escribe de manera que el procesador ejecuta instrucciones que le dan control directo de la operación de E/S, incluyendo comprobar el estado del dispositivo, enviar un mandato de lectura o de escritura, y transferir los datos. Por tanto, el juego de instrucciones incluye instrucciones de E/S de las siguientes categorías:

- Control. Utilizadas para activar un dispositivo externo y especificarle qué debe hacer. Por ejemplo, se le puede indicar a una unidad de cinta magnética que se rebobine o avance un registro.
- Estado. Utilizadas para comprobar diversas condiciones de estado asociadas a un módulo de E/S y sus periféricos.
- Transferencia. Utilizadas para leer y/o escribir datos entre los registros del procesador y los dispositivos externos.

E/S dirigidas por interrupciones

El problema de la E/S programada es que el procesador tiene que esperar mucho tiempo hasta que el módulo de E/S correspondiente esté listo para la recepción o la transmisión de más datos. Una alternativa es que el procesador genere un mandato de E/S para un módulo y, acto seguido, continúe realizando algún otro trabajo útil. El módulo de E/S interrumpirá más tarde al procesado para solicitar su servicio cuando esté listo para intercambiar datos con el mismo. El procesador ejecutará la transferencia de datos, como antes, y después reanudará el procesamiento previo.

Acceso Directo a Memoria

La E/S dirigida por interrupciones, aunque más eficiente que la E/S programada simple, todavía requiere la intervención activa del procesador para transferir datos entre la memoria y un módulo de E/S, ya que cualquier transferencia de datos debe atravesar un camino a través del procesador.

1. La tasa de transferencia de E/S está limitada por la velocidad con la que el procesador puede comprobar el estado de un dispositivo y ofrecerle servicio.
2. El procesador está involucrado en la gestión de una transferencia de E/S; se deben ejecutar varias instrucciones por cada transferencia de E/S.

Cuando el procesador desea leer o escribir un bloque de datos, genera un mandato al módulo de DMA, enviándole la siguiente información:

- Si se trata de una lectura o de una escritura.
- La dirección del dispositivo de E/S involucrado.
- La posición inicial de memoria en la que se desea leer los datos o donde se quieren escribir.
- El número de palabras que se pretende leer o escribir.

A continuación, el procesador continúa con otro trabajo. Ha delegado esta operación de E/S al módulo de DMA, que se ocupará de la misma. El módulo de DMA transferirá el bloque completo de datos, palabra a palabra, hacia la memoria o desde ella sin pasar a través del procesador. Por tanto, el procesador solamente está involucrado al principio y al final de la transferencia.

El módulo de DMA necesita tomar el control del bus para transferir datos hacia la memoria o desde ella. Debido a esta competencia en el uso del bus, puede haber veces en las que el procesador necesita el bus y debe esperar al módulo de DMA. Nótese que esto no es una interrupción; el procesador no salva un contexto y pasa a hacer otra cosa. En su lugar, el procesador se detiene durante un ciclo de bus (el tiempo que se tarda en transferir una palabra a través del bus). El efecto global es causar que el procesador ejecute más lentamente durante una transferencia de DMA en el caso de que el procesador requiera acceso al bus. Sin embargo, para una transferencia de E/S de

múltiples palabras, el DMA es mucho más eficiente que la E/S dirigida por interrupciones o la programada.

2. Introducción a los Sistemas Operativos

2.1 Objetivos y funciones de los sistemas operativos

Un sistema operativo es un programa que controla la ejecución de aplicaciones y programas y que actúa como interfaz entre las aplicaciones y el hardware del computador. Se puede considerar que un sistema operativo tiene los siguientes tres objetivos:

- Facilidad de uso. Un sistema operativo facilita el uso de un computador.
- Eficiencia. Un sistema operativo permite que los recursos de un sistema de computación se puedan utilizar de una manera eficiente.
- Capacidad para evolucionar. Un sistema operativo se debe construir de tal forma que se puedan desarrollar, probar e introducir nuevas funciones en el sistema sin interferir con su servicio.

Entre sus servicios, los SO ofrecen el desarrollo de programas, ejecución de programas, acceso a dispositivos de E/S, acceso controlado a los ficheros, acceso al sistema, detección y respuesta frente a errores y contabilidad, entre otros.

El sistema operativo como gestor de recursos

Un computador es un conjunto de recursos que se utilizan para el transporte, almacenamiento y procesamiento de los datos, así como para llevar a cabo el control de estas funciones. El sistema operativo se encarga de gestionar estos recursos.

El sistema operativo dirige al procesador en el uso de los otros recursos del sistema y en la temporización de la ejecución de otros programas. No obstante, para que el procesador pueda realizar esto, el sistema operativo debe dejar paso a la ejecución de otros programas. Por tanto, el sistema operativo deja el control para que el procesador pueda realizar trabajo «útil» y de nuevo retoma el control para permitir al procesador que realice la siguiente pieza de trabajo.

Una porción del sistema operativo se encuentra en la memoria principal. Esto incluye el kernel, o núcleo, que contiene las funciones del sistema operativo más frecuentemente utilizadas y, en cierto momento, otras porciones del sistema operativo actualmente en uso. El resto de la memoria principal contiene programas y datos de usuario. La asignación de este recurso (memoria principal) es controlada de forma conjunta por el sistema operativo y el hardware de gestión de memoria del procesador, como se verá. El sistema operativo decide cuándo un programa en ejecución puede utilizar un dispositivo de E/S y controla el acceso y uso de los ficheros. El procesador es también un

recurso, y el sistema operativo debe determinar cuánto tiempo de procesador debe asignarse a la ejecución de un programa de usuario particular. En el caso de un sistema multiprocesador, esta decisión debe ser tomada por todos los procesadores.

2.2 La evolución de los sistemas operativos

Procesamiento en serie

El programador interaccionaba directamente con el hardware del computador; no existía ningún sistema operativo. Los programas en código máquina se cargaban a través del dispositivo de entrada (por ejemplo, un lector de tarjetas). Si un error provocaba la parada del programa, las luces indicaban la condición de error.

Sistemas en lotes sencillos

La idea central bajo el esquema de procesamiento en lotes sencillo es el uso de una pieza de software denomina monitor. Con este tipo de sistema operativo, el usuario no tiene que acceder directamente a la máquina. En su lugar, el usuario envía un trabajo a través de una tarjeta o cinta al operador del computador, que crea un sistema por lotes con todos los trabajos enviados y coloca la secuencia de trabajos en el dispositivo de entrada, para que lo utilice el monitor. Cuando un programa finaliza su procesamiento, devuelve el control al monitor, punto en el cual dicho monitor comienza la carga del siguiente programa.

Sistemas en lotes multiprogramados

El procesador se encuentra frecuentemente ocioso, incluso con el secuenciamiento de trabajos automático que proporciona un sistema operativo en lotes simple. El problema consiste en que los dispositivos de E/S son lentos comparados con el procesador. Como solución cuando un trabajo necesita esperar por la E/S, se puede asignar el procesador al otro trabajo, que probablemente no esté esperando por una operación de E/S. Este enfoque se conoce como multiprogramación o multitarea. Es el tema central de los sistemas operativos modernos.

Sistemas de tiempo compartido

Del mismo modo que la multiprogramación permite al procesador gestionar múltiples trabajos en lotes en un determinado tiempo, la multiprogramación también se puede utilizar para gestionar múltiples trabajos interactivos. En este último caso, la técnica se denomina tiempo compartido, porque se comparte el tiempo de procesador entre múltiples usuarios. En un sistema de tiempo compartido, múltiples usuarios acceden simultáneamente al sistema a través de terminales, siendo el sistema operativo el encargado de entrelazar la ejecución de cada programa de usuario en pequeños intervalos de tiempo o cuantos de computación.

2.4 Algunos conceptos de los SO modernos

Arquitectura micronúcleo o microkernel

Hasta hace relativamente poco tiempo, la mayoría de los sistemas operativos estaban formados por un gran núcleo monolítico. Estos grandes núcleos proporcionan la mayoría de las funcionalidades consideradas propias del sistema operativo, incluyendo la planificación, los sistemas de ficheros, las redes, los controladores de dispositivos, la gestión de memoria y otras funciones.

Una arquitectura micronúcleo asigna sólo unas pocas funciones esenciales al núcleo, incluyendo los espacios de almacenamiento, comunicación entre procesos (IPC), y la planificación básica. Ciertos procesos proporcionan otros servicios del sistema operativo, algunas veces denominados servidores, que ejecutan en modo usuario y son tratados como cualquier otra aplicación por el micronúcleo. Esta técnica desacopla el núcleo y el desarrollo del servidor. Los servidores pueden configurarse para aplicaciones específicas o para determinados requisitos del entorno. La técnica micronúcleo simplifica la implementación, proporciona flexibilidad y se adapta perfectamente a un entorno distribuido. En esencia, un micronúcleo interactúa con procesos locales y remotos del servidor de la misma forma, facilitando la construcción de los sistemas distribuidos.

Multihilo

Multitheading es una técnica en la cual un proceso, ejecutando una aplicación, se divide en una serie de hilos o threads que pueden ejecutar concurrentemente. Se pueden hacer las siguientes distinciones:

- Thread o hilo. Se trata de una unidad de trabajo. Incluye el contexto del procesador (que contiene el contador del programa y el puntero de pila) y su propia área de datos para una pila (para posibilitar el salto a subrutinas). Un hilo se ejecuta secuencialmente y se puede interrumpir de forma que el procesador pueda dar paso a otro hilo.
- Proceso. Es una colección de uno o más hilos y sus recursos de sistema asociados (como la memoria, conteniendo tanto código, como datos, ficheros abiertos y dispositivos). Esto corresponde al concepto de programa en ejecución. Dividiendo una sola aplicación en múltiples hilos, el programador tiene gran control sobre la modularidad de las aplicaciones y la temporización de los eventos relacionados con la aplicación.

La técnica multithreading es útil para las aplicaciones que llevan a cabo un número de tareas esencialmente independientes que no necesitan ser serializadas.

Multiprocesamiento simétrico

Hasta hace poco tiempo, los computadores personales y estaciones de trabajo virtualmente de un único usuario contenían un único procesador de propósito general. A medida que la demanda de rendimiento se incrementa y el coste de los microprocesadores continúa cayendo, los fabricantes han introducido en el mercado computadores con múltiples procesadores. Para lograr mayor eficiencia y fiabilidad, una técnica consiste en emplear multiprocesamiento simétrico (SMP: Symmetric Multi-Processing), un término que se refiere a la arquitectura hardware del computador y también al comportamiento del sistema operativo que explota dicha arquitectura. Se puede definir un

multiprocesador simétrico como un sistema de computación aislado con las siguientes características:

1. Tiene múltiples procesadores.
2. Estos procesadores comparten las mismas utilidades de memoria principal y de E/S, interconectadas por un bus de comunicación u otro esquema de conexión interna.
3. Todos los procesadores pueden realizar las mismas funciones.

El sistema operativo de un SMP planifica procesos o hilos a través de todos los procesadores. El sistema operativo debe proporcionar herramientas y funciones para explotar el paralelismo en un sistema SMP.

La técnica multithreading y SMP son frecuentemente analizados juntos, pero son dos utilidades independientes. Una máquina SMP es útil incluso para procesos que no contienen hilos, porque varios procesos pueden ejecutar en paralelo. Sin embargo, ambas utilidades se complementan y se pueden utilizar de forma conjunta efectivamente. Una característica atractiva de un SMP es que la existencia de múltiples procesadores es transparente al usuario. El sistema operativo se encarga de planificar los hilos o procesos en procesadores individuales y de la sincronización entre los procesadores.

Sistemas distribuidos

Un sistema operativo distribuido proporciona la ilusión de un solo espacio de memoria principal y un solo espacio de memoria secundario, más otras utilidades de acceso unificadas, como un sistema de ficheros distribuido. Aunque los clusters se están volviendo cada día más populares, y hay muchos productos para clusters en el mercado, el estado del arte de los sistemas distribuidos está retrasado con respecto a los monoprocesadores y sistemas operativos SMP.

Diseño orientado a objetos

El diseño orientado a objetos introduce una disciplina al proceso de añadir extensiones modulares a un pequeño núcleo. A nivel del sistema operativo, una estructura basada en objetos permite a los programadores personalizar un sistema operativo sin eliminar la integridad del sistema. La orientación a objetos también facilita el desarrollo de herramientas distribuidas y sistemas operativos distribuidos.

2.5 Windows

Arquitectura

Su estructura modular da a Windows una considerable flexibilidad. Se ha diseñado para ejecutar en una variedad de plataformas hardware y da soporte a aplicaciones escritas para una gran cantidad de sistemas operativos. Windows separa el software orientado a aplicación del software del sistema operativo. La segunda parte, que incluye el sistema ejecutivo, el núcleo o kernel y la capa de abstracción del hardware, ejecuta en modo núcleo. El software que ejecuta en modo núcleo tiene

acceso a los datos del sistema y al hardware. El resto del software, que ejecuta en modo usuario, tiene un acceso limitado a los datos del sistema.

Organización del sistema operativo

Windows no tiene una arquitectura micronúcleo pura, sino lo que Microsoft denomina arquitectura micronúcleo modificada. Como en el caso de las arquitecturas micronúcleo puras, Windows es muy modular. Cada función del sistema se gestiona mediante un único componente del sistema operativo. El resto del sistema operativo y todas las aplicaciones acceden a dicha función a través del componente responsable y utilizando una interfaz estándar. Sólo se puede acceder a los datos del sistema claves mediante la función apropiada. En principio, se puede borrar, actualizar o reemplazar cualquier módulo sin reescribir el sistema completo o su interfaz de programa de aplicación (API). Sin embargo, a diferencia de un sistema micronúcleo puro, Windows se configura de forma que muchas de las funciones del sistema externas al micronúcleo ejecutan en modo núcleo.

Los componentes en modo núcleo son los siguientes:

- Sistema ejecutivo. Contiene los servicios básicos del sistema operativo, como la gestión de memoria, la gestión de procesos e hilos, seguridad, E/S y comunicación entre procesos.
- Núcleo. Está formado por los componentes más fundamentales del sistema operativo. El núcleo gestiona la planificación de hilos, el intercambio de procesos, las excepciones, el manejo de interrupciones y la sincronización de multiprocesadores. A diferencia del resto del sistema ejecutivo y el nivel de usuario, el código del núcleo no se ejecuta en hilos. Por tanto, es la única parte del sistema operativo que no es expulsable o paginable.
- Capa de abstracción de hardware (HAL: Hardware Abstraction Layer). Realiza una proyección entre mandatos y respuestas hardware genéricos y aquéllos que son propios de una plataforma específica. Aísla el sistema operativo de las diferencias de hardware específicas de la plataforma. El HAL hace que el bus de sistema, el controlador de acceso a memoria directa (DMA), el controlador de interrupciones, los temporizadores de sistema y los módulos de memoria de cada máquina parezcan los mismos al núcleo. También entrega el soporte necesario para multiprocesamiento simétrico (SMP), explicado anteriormente.
- Controladores de dispositivo. Incluye tanto sistemas de ficheros como controladores de dispositivos hardware que traducen funciones de E/S de usuario en peticiones específicas a dispositivos hardware de E/S.
- Gestión de ventanas y sistemas gráficos. Implementa las funciones de la interfaz gráfica de usuario (GUI), tales como la gestión de ventanas, los controles de la interfaz de usuario y el dibujo.

Procesos en modo usuario

Hay cuatro tipos básicos de procesos en modo usuario en Windows:

- Procesos de sistema especiales. Incluye servicios no proporcionados como parte del sistema operativo Windows, como el proceso de inicio y el gestor de sesiones.

- Procesos de servicio. Otros servicios de Windows como por ejemplo, el registro de eventos.
- Subsistemas de entorno. Expone los servicios nativos de Windows a las aplicaciones de usuario y por tanto, proporciona un entorno o personalidad de sistema operativo. Los subsistemas soportados son Win32, Posix y OS/2. Cada subsistema de entorno incluye bibliotecas de enlace dinámico (Dynamic Link Libraries, DLL), que convierten las llamadas de la aplicación de usuario a llamadas Windows.
- Aplicaciones de usuario. Pueden ser de cinco tipos: Win32, Posix, OS/2, Windows 3.1 o MS-DOS.

Modelo Cliente-Servidor

El sistema ejecutivo, los subsistemas protegidos y las aplicaciones se estructuran de acuerdo al modelo de computación cliente/servidor, que es un modelo común para la computación distribuida. Esta misma arquitectura se puede adoptar para el uso interno de un solo sistema, como es el caso de Windows. Cada subsistema de entorno y subsistema del servicio ejecutivo se implementa como uno o más procesos. Cada proceso espera la solicitud de un cliente por uno de sus servicios (por ejemplo, servicios de memoria, servicios de creación de procesos o servicios de planificación de procesadores). Un cliente, que puede ser un programa u otro módulo del sistema operativo, solicita un servicio a través del envío de un mensaje. El mensaje se encamina a través del sistema ejecutivo al servidor apropiado. El servidor lleva a cabo la operación requerida y devuelve los resultados o la información de estado por medio de otro mensaje, que se encamina de vuelta al cliente mediante el servicio ejecutivo.

Hilos y SMP

Dos características importantes de Windows son el soporte que da a los hilos y a SMP:

- Las rutinas del sistema operativo se pueden ejecutar en cualquier procesador disponible, y diferentes rutinas se pueden ejecutar simultáneamente en diferentes procesadores.
- Windows permite el uso de múltiple hilos de ejecución dentro de un único proceso. Múltiples hilos dentro del mismo proceso se pueden ejecutar en diferentes procesadores simultáneamente.
- Los procesos de servidor pueden utilizar múltiples hilos para procesar peticiones de más de un cliente simultáneamente.
- Windows proporciona mecanismos para compartir datos y recursos entre procesos y capacidades flexibles de comunicación entre procesos.

Objetos en Windows

Windows se apoya enormemente en los conceptos del diseño orientado a objetos. Este enfoque facilita la compartición de recursos y datos entre los procesos y la protección de recursos frente al acceso no autorizado.

2.8 Linux

La mayoría de los núcleos de linux son monolíticos. Todos los componentes funcionales del núcleo tienen acceso a todas las estructuras internas de datos y rutinas.

Linux está estructurado como una colección de módulos, algunos de los cuales pueden cargarse y descargarse automáticamente bajo demanda. Estos bloques relativamente independientes se denominan módulos cargables [GOYE99]. Esencialmente, un módulo es un fichero cuyo código puede enlazarse y desenlazarse con el núcleo en tiempo real.

Por tanto, aunque Linux se puede considerar monolítico, su estructura modular elimina algunas de las dificultades para desarrollar y evolucionar el núcleo.

Los módulos cargables de Linux tienen dos características importantes:

- Enlace dinámico. Un módulo de núcleo puede cargarse y enlazarse al núcleo mientras el núcleo está en memoria y ejecutándose. Un módulo también puede desenlazarse y eliminarse de la memoria en cualquier momento.
- Módulos apilables. Los módulos se gestionan como una jerarquía. Los módulos individuales sirven como bibliotecas cuando los módulos cliente los referencian desde la parte superior de la jerarquía, y actúan como clientes cuando referencian a módulos de la parte inferior de la jerarquía.

Componentes del Núcleo

- Señales. El núcleo utiliza las señales para llamar a un proceso. Por ejemplo, las señales se utilizan para notificar ciertos fallos a un proceso como por ejemplo, la división por cero.
- Llamadas al sistema. La llamada al sistema es la forma en la cual un proceso requiere un servicio de núcleo específico. Hay varios cientos de llamadas al sistema, que pueden agruparse básicamente en seis categorías: sistema de ficheros, proceso, planificación, comunicación entre procesos, socket (red) y misceláneos. La Tabla 2.7 define unos pocos ejemplos de cada categoría.
- Procesos y planificador. Crea, gestiona y planifica procesos.
- Memoria virtual. Asigna y gestiona la memoria virtual para los procesos.
- Sistemas de ficheros. Proporciona un espacio de nombres global y jerárquico para los ficheros, directorios y otros objetos relacionados con los ficheros. Además, proporciona las funciones del sistema de ficheros.
- Protocolos de red. Da soporte a la interfaz Socket para los usuarios, utilizando la pila de protocolos TCP/IP.
- Controladores de dispositivo tipo carácter. Gestiona los dispositivos que requiere el núcleo para enviar o recibir datos un byte cada vez, como los terminales, los módems y las impresoras.

- Controladores de dispositivo tipo bloque. Gestiona dispositivos que leen y escriben datos en bloques, tal como varias formas de memoria secundaria (discos magnéticos, CDROM, etc.).
- Controladores de dispositivo de red. Gestiona las tarjetas de interfaz de red y los puertos de comunicación que permiten las conexiones a la red, tal como los puentes y los encaminadores.
- Traps y fallos. Gestiona los traps y fallos generados por la CPU, como los fallos de memoria.
- Memoria física. Gestiona el conjunto de marcos de páginas de memoria real y asigna las páginas de memoria virtual.
- Interrupciones. Gestiona las interrupciones de los dispositivos periféricos.

3. Procesos

3.1 Definición

Surgen diversas definiciones del término proceso, incluyendo:

- **Un programa en ejecución**.
- Una instancia de un programa ejecutado en un computador.
- La entidad que se puede asignar y ejecutar en un procesador.
- Una unidad de actividad que se caracteriza por la ejecución de una secuencia de instrucciones, un estado actual, y un conjunto de recursos del sistema asociados.

También se puede pensar en un proceso como en una entidad que consiste en un número de elementos. Los dos elementos esenciales serían el código de programa (que puede compartirse con otros procesos que estén ejecutando el mismo programa) y un conjunto de datos asociados a dicho código. Supongamos que el procesador comienza a ejecutar este código de programa, y que nos referiremos a esta entidad en ejecución como un proceso. En cualquier instante puntual del tiempo, mientras el proceso está en ejecución, este proceso se puede caracterizar por una serie de elementos, incluyendo los siguientes:

- Identificador. Un identificador único asociado a este proceso, para distinguirlo del resto de procesos.
- Estado. Si el proceso está actualmente corriendo, está en el estado en ejecución.
- Prioridad: Nivel de prioridad relativo al resto de procesos.
- Contador de programa. La dirección de la siguiente instrucción del programa que se ejecutará.
- Punteros a memoria. Incluye los punteros al código de programa y los datos asociados a dicho proceso, además de cualquier bloque de memoria compartido con otros procesos.
- Datos de contexto. Estos son datos que están presenten en los registros del procesador cuando el proceso está corriendo.

- Información de estado de E/S. Incluye las peticiones de E/S pendientes, dispositivos de E/S (por ejemplo, una unidad de cinta) asignados a dicho proceso, una lista de los ficheros en uso por el mismo, etc.
- Información de auditoría. Puede incluir la cantidad de tiempo de procesador y de tiempo de reloj utilizados, así como los límites de tiempo, registros contables, etc.
- Valores de registro de CPU. Se utilizan también en el cambio de contexto.
- Espacio de direcciones de memoria.
- Lista de recursos asignados (incluyendo descriptores de archivos y sockets abiertos).
- Estadísticas del proceso.
- Datos del propietario (owner).
- Permisos asignados.

La información de la lista anterior se almacena en una estructura de datos, que se suele llamar **bloque de control de proceso** (process control block), que el sistema operativo crea y gestiona. El punto más significativo en relación al bloque de control de proceso, o PCB, es que contiene suficiente información de forma que es posible interrumpir el proceso cuando está corriendo y posteriormente restaurar su estado de ejecución como si no hubiera habido interrupción alguna. El PCB es la herramienta clave que permite al sistema operativo dar soporte a múltiples procesos y proporcionar multiprogramación. Cuando un proceso se interrumpe, los valores actuales del contador de programa y los registros del procesador (datos de contexto) se guardan en los campos correspondientes del PCB y el estado del proceso se cambia a cualquier otro valor, como bloqueado o listo (descritos a continuación). El sistema operativo es libre ahora para poner otro proceso en estado de ejecución. El contador de programa y los datos de contexto se recuperan y cargan en los registros del procesador y este proceso comienza a correr.
De esta forma, se puede decir que un proceso está compuesto del código de programa y los datos asociados, además del bloque de control de proceso o BCP

3.2 Estados de procesos

Desde el punto de vista del procesador, él ejecuta instrucciones de su repertorio de instrucciones en una secuencia dictada por el cambio de los valores del registro contador de programa. A lo largo del tiempo, el contador de programa puede apuntar al código de diferentes programas que son parte de diferentes procesos. Desde el punto de vista de un programa individual, su ejecución implica una secuencia de instrucciones dentro de dicho programa.
Se puede caracterizar el comportamiento de un determinado proceso, listando la secuencia de instrucciones que se ejecutan para dicho proceso. A esta lista se la denomina **traza del proceso**. Se puede caracterizar el comportamiento de un procesador mostrando cómo las trazas de varios procesos se entrelazan.
De manera adicional, existe un pequeño programa activador (**dispatcher**) que intercambia el procesador de un proceso a otro.

Creación y terminación de procesos

Creación de un proceso. Cuando se va a añadir un nuevo proceso a aquellos que se están gestionando en un determinado momento, el sistema operativo construye las estructuras de datos que se usan para manejar el proceso y reserva el espacio de direcciones en memoria principal para el proceso.

Estas acciones constituyen la creación de un nuevo proceso.

Nuevo proceso de lotes	El sistema operativo dispone de un flujo de control de lotes de trabajos, habitualmente una cinta un disco. Cuando el sistema operativo está listo para procesar un nuevo trabajo, leerá la siguiente secuencia de mandatos de control de trabajos.
Sesión interactiva	Un usuario desde un terminal entra en el sistema.
Creado por el sistema operativo para proporcionar un servicio	El sistema operativo puede crear un proceso para realizar una función en representación de un programa de usuario, sin que el usuario tenga que esperar (por ejemplo, un proceso para controlar la impresión).
Creado por un proceso existente	Por motivos de modularidad o para explotar el paralelismo, un programa de usuario puede ordenar la creación de un número de procesos.

Cuando un proceso lanza otro, al primero se le denomina proceso padre, y al proceso creado se le denomina proceso hijo.

Cuando un proceso crea un subproceso, este puede obtener sus recursos directamente del sistema operativo o puede estar restringido a un subconjunto de recursos del padre. El padre puede repartir sus recursos entre sus hijos o puede compartir sólo algunos (como la memoria o los archivos). Restringir un proceso hijo a un subconjunto de recursos del padre evita que un proceso pueda sobrecargar el sistema.

Cuando un proceso crea otro proceso, existen dos posibilidades en términos de ejecución:

1. El padre continúa ejecutándose concurrentemente con su hijo.
2. El padre espera hasta que alguno o todos los hijos han terminado de ejecutarse.

También existen dos posibilidades en función del espacio de direcciones del nuevo proceso:

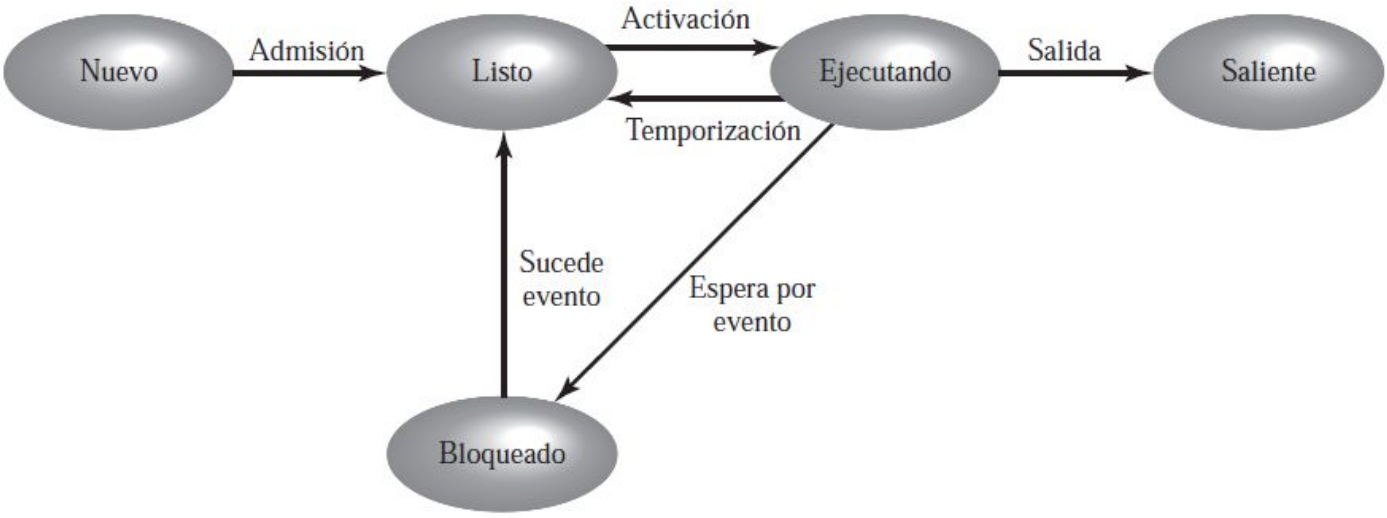
1. El proceso hijo es un duplicado del proceso padre (usa el mismo programa y los mismos datos que el padre

El proceso hijo carga un nuevo programa

Terminación de procesos. Todo sistema debe proporcionar los mecanismos mediante los cuales un proceso indica su finalización, o que ha completado su tarea.

Finalización normal	El proceso ejecuta una llamada al sistema operativo para indicar que ha completado su ejecución
Límite de tiempo excedido	El proceso ha ejecutado más tiempo del especificado en un límite máximo. Existen varias posibilidades para medir dicho tiempo. Estas incluyen el tiempo total utilizado, el tiempo utilizado únicamente en ejecución, y, en el caso de procesos interactivos, la cantidad de tiempo desde que el usuario realizó la última entrada.
Memoria no disponible	El proceso requiere más memoria de la que el sistema puede proporcionar.
Violaciones de frontera	El proceso trata de acceder a una posición de memoria a la cual no tiene acceso permitido.
Error de protección	El proceso trata de usar un recurso, por ejemplo un fichero, al que no tiene permitido acceder, o trata de utilizarlo de una forma no apropia, por ejemplo, escribiendo en un fichero de sólo lectura.
Error aritmético	El proceso trata de realizar una operación de cálculo no permitida, tal como una división por 0, o trata de almacenar números mayores de los que la representación hardware puede codificar.
Límite de tiempo	El proceso ha esperado más tiempo que el especificado en un valor máximo para que se cumpla un determinado evento.
Fallo de E/S	Se ha producido un error durante una operación de entrada o salida, por ejemplo la imposibilidad de encontrar un fichero, fallo en la lectura o escritura después de un límite máximo de intentos (cuando, por ejemplo, se encuentra un área defectuosa en una cinta), o una operación inválida (la lectura de una impresora en línea).
Instrucción no válida	El proceso intenta ejecutar una instrucción inexistente (habitualmente el resultado de un salto a un área de datos y el intento de ejecutar dichos datos).
Instrucción privilegiada	El proceso intenta utilizar una instrucción reservada al sistema operativo.
Uso inapropiado de datos	Una porción de datos es de tipo erróneo o no se encuentra inicializada.
Intervención del operador por el sistema operativo	Por alguna razón, el operador o el sistema operativo ha finalizado el proceso (por ejemplo, se ha dado una condición de interbloqueo).
Terminación del proceso padre	Cuando un proceso padre termina, el sistema operativo puede automáticamente finalizar todos los procesos hijos descendientes de dicho padre.
Solicitud del proceso padre	Un proceso padre habitualmente tiene autoridad para finalizar sus propios procesos descendientes.

Modelo de proceso de 5 estados



- Ejecutando. El proceso está actualmente en ejecución.
- Listo. Un proceso que se prepara para ejecutar cuando tenga oportunidad.
- Bloqueado. Un proceso que no puede ejecutar hasta que se cumpla un evento determinado o se complete una operación E/S.
- Nuevo. Un proceso que se acaba de crear y que aún no ha sido admitido en el grupo de procesos ejecutables por el sistema operativo. Típicamente, se trata de un nuevo proceso que no ha sido cargado en memoria principal, aunque su bloque de control de proceso (BCP) si ha sido Creado.
- Saliente. Un proceso que ha sido liberado del grupo de procesos ejecutables por el sistema operativo, debido a que ha sido detenido o que ha sido abortado por alguna razón.

Los estados Nuevo y Saliente son útiles para construir la gestión de procesos. El estado Nuevo se corresponde con un proceso que acaba de ser definido.

Pases entre estados:

- Null > Nuevo. Se crea un nuevo proceso para ejecutar un programa.
- Nuevo > Listo. El sistema operativo mueve a un proceso del estado nuevo al estado listo cuando éste se encuentre preparado para ejecutar un nuevo proceso. La mayoría de sistemas fijan un límite basado en el número de procesos existentes o la cantidad de memoria virtual que se podrá utilizar por parte de los procesos existentes. Este límite asegura que no haya demasiados procesos activos y que se degrade el rendimiento sistema.
- Listo > Ejecutando. Cuando llega el momento de seleccionar un nuevo proceso para ejecutar, el sistema operativo selecciona uno los procesos que se encuentre en el estado Listo. Esta es una tarea la lleva a cabo el planificador.

- Ejecutando > Saliente. el proceso actual en ejecución se finaliza por parte del sistema operativo tanto si el proceso indica que ha completado su ejecución como si éste se aborta.
- Ejecutando > Listo. La razón más habitual para esta transición es que el proceso en ejecución haya alcanzado el máximo tiempo posible de ejecución de forma ininterrumpida; prácticamente todos los sistemas operativos multiprogramados imponen este tipo de restricción de tiempo. Existen otras posibles causas alternativas para esta transición, que no están incluidas en todos los sistemas operativos. Es de particular importancia el caso en el cual el sistema operativo asigna diferentes niveles de prioridad a diferentes procesos.

  Por ejemplo, que el proceso A está ejecutando a un determinado nivel de prioridad, y el proceso B, a un nivel de prioridad mayor, y que se encuentra bloqueado. Si el sistema operativo se da cuenta de que se produce un evento al cual el proceso B está esperando, moverá el proceso B al estado de Listo. Esto puede interrumpir al proceso A y poner en ejecución al proceso B. Decimos, en este caso, que el sistema operativo ha expulsado al proceso A. Adicionalmente, un proceso puede voluntariamente dejar de utilizar el procesador. Un ejemplo son los procesos que realiza alguna función de auditoría o de mantenimiento de forma periódica.
- Ejecutando > Bloqueado. Un proceso se pone en el estado Bloqueado si solicita algo por lo cual debe esperar. Una solicitud al sistema operativo se realiza habitualmente por medio de una llamada al sistema; esto es, una llamada del proceso en ejecución a un procedimiento que es parte del código del sistema operativo.

  Por ejemplo, un proceso ha solicitado un servicio que el sistema operativo no puede realizar en ese momento. Puede solicitar un recurso, como por ejemplo un fichero o una sección compartida de memoria virtual, que no está inmediatamente disponible. Cuando un proceso quiere iniciar una acción, tal como una operación de E/S, que debe completarse antes de que el proceso continúe. Cuando un proceso se comunica con otro, un proceso puede bloquearse mientras está esperando a que otro proceso le proporcione datos o esperando un mensaje de ese otro proceso.
- Bloqueado > Listo. Un proceso en estado Bloqueado se mueve al estado Listo cuando sucede el evento por el cual estaba esperando.
- Listo > Saliente. Por claridad, esta transición no se muestra en el diagrama de estados. En algunos sistemas, un padre puede terminar la ejecución de un proceso hijo en cualquier momento. También, si el padre termina, todos los procesos hijos asociados con dicho padre pueden finalizarse.
- Bloqueado > Saliente. Se aplican los comentarios indicados en el caso anterior.

Procesos suspendidos

Vamos a suponer un sistema que no utiliza memoria virtual. Cada proceso que se ejecuta debe cargarse completamente en memoria principal. Recuérdese que la razón de toda esta compleja

maquinaria es que las operaciones de E/S son mucho más lentas que los procesos de cómputo y, por tanto, el procesador en un sistema monoprogramado estaría ocioso la mayor parte del tiempo.
La diferencia de velocidad entre el procesador y la E/S es tal que sería muy habitual que todos los procesos en memoria se encontrasen a esperas de dichas operaciones. Por tanto, incluso en un sistema multiprogramado, el procesador puede estar ocioso la mayor parte del tiempo.

La solución es el **swapping** (memoria de intercambio), que implica mover parte o todo el proceso de memoria principal al disco. Cuando ninguno de los procesos en memoria principal se encuentra en estado Listo, el sistema operativo intercambia uno de los procesos bloqueados a disco, en la cola de Suspendidos. Esta es una lista de procesos existentes que han sido temporalmente expulsados de la memoria principal, o suspendidos. El sistema operativo trae otro proceso de la cola de Suspendidos o responde a una solicitud de un nuevo proceso. La ejecución continúa con los nuevos procesos que han llegado.

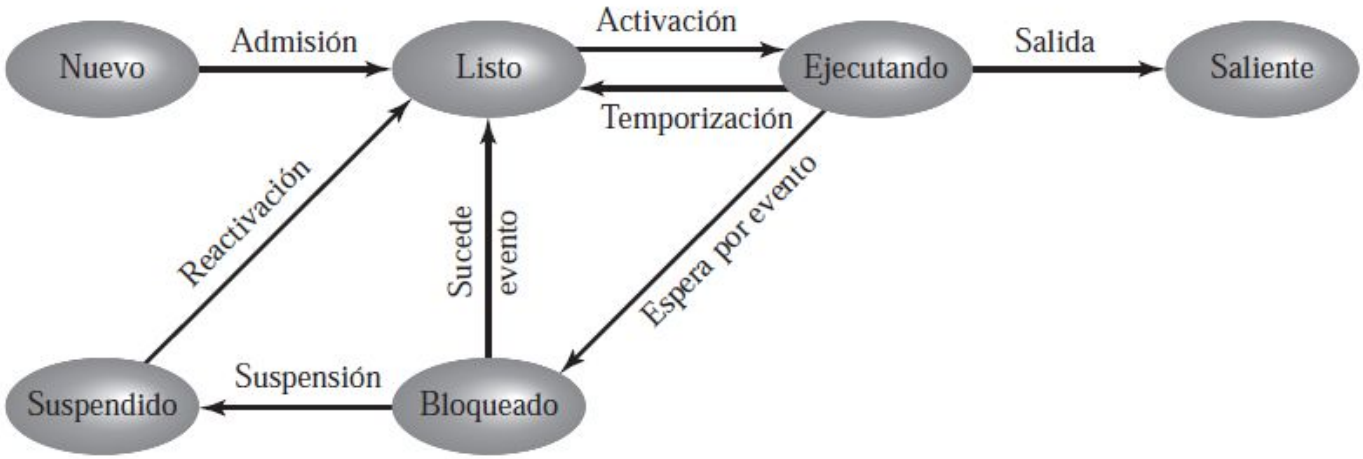
El swapping, sin embargo, es una operación de E/S, y por tanto existe el riesgo potencial de hacer que el problema empeore. Pero debido a que la E/S en disco es habitualmente más rápida que la E/S sobre otros sistemas (por ejemplo, comparado con cinta o impresora), el swapping habitualmente mejora el rendimiento del sistema.

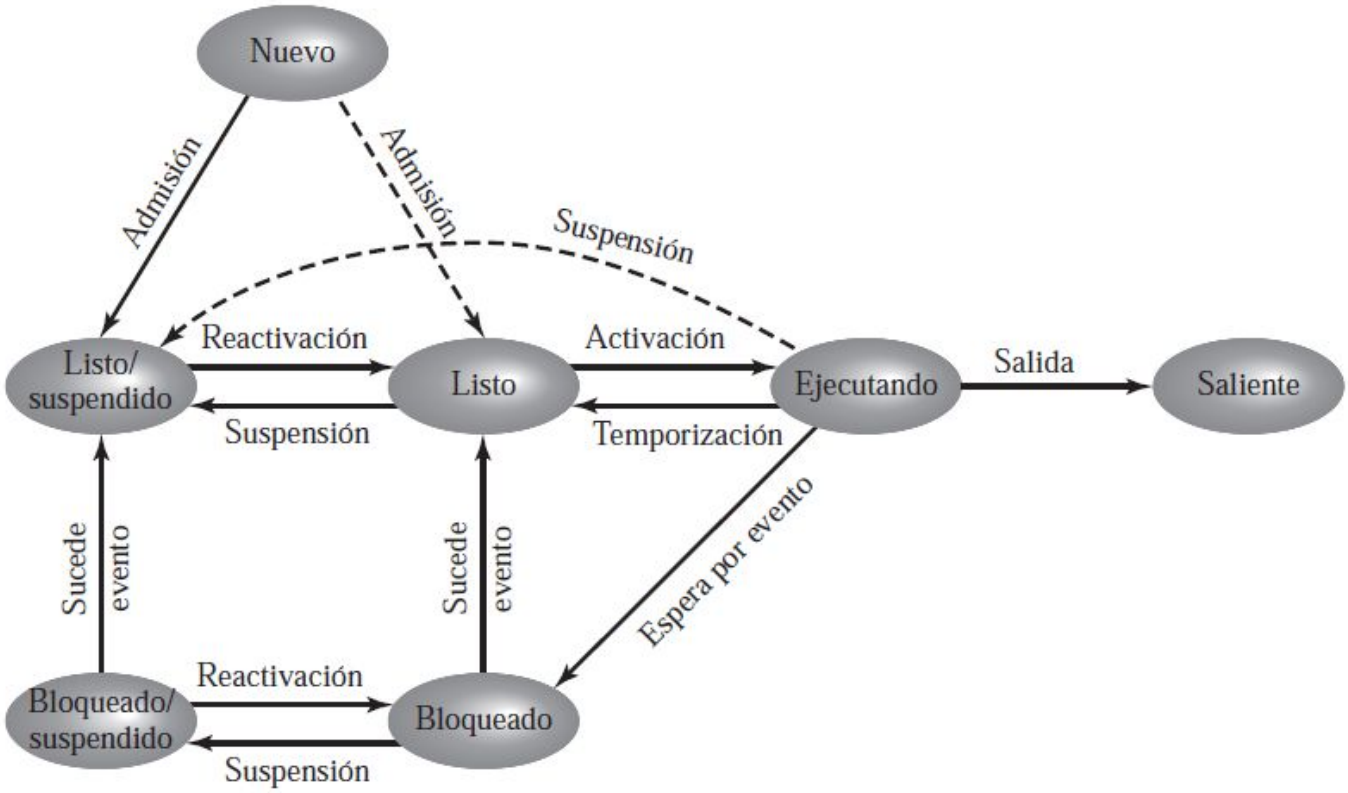
Cuando todos los procesos en memoria principal se encuentran en estado Bloqueado, el sistema operativo puede suspender un proceso poniéndolo en el estado Suspendido y transfiriéndolo a disco. El espacio que se libera en memoria principal puede usarse para traer a otro proceso.

Cuando el sistema operativo ha realizado la operación de swap (transferencia a disco de un proceso), tiene dos opciones para seleccionar un nuevo proceso para traerlo a memoria principal: puede admitir un nuevo proceso que se haya creado o puede traer un proceso que anteriormente se encontrase en estado de Suspendido. Parece que sería preferible traer un proceso que anteriormente estuviese suspendido, para proporcionar dicho servicio en lugar de incrementar la carga total del sistema.
Pero, esta línea de razonamiento presenta una dificultad: todos los procesos que fueron suspendidos se encontraban previamente en el estado de Bloqueado en el momento de su suspensión. Claramente no sería bueno traer un proceso bloqueado de nuevo a memoria porque podría no encontrarse todavía listo para la ejecución. Se debe reconocer, sin embargo, que todos los procesos en estado Suspendido estaban originalmente en estado Bloqueado, en espera de un evento en particular. Cuando el evento sucede, el proceso no está Bloqueado y está potencialmente disponible para su ejecución.

De esta forma, necesitamos replantear este aspecto del diseño. Hay dos conceptos independientes aquí: si un proceso está esperando a un evento (Bloqueado o no) y si un proceso está transferido de memoria a disco (suspendido o no). Para describir estas combinaciones de 2 ¥ 2 necesitamos cuatro estados:

- Listo. El proceso está en memoria principal disponible para su ejecución.
- Bloqueado. El proceso está en memoria principal y esperando un evento.
- Bloqueado/Suspendido. El proceso está en almacenamiento secundario y esperando un evento.
- Listo/Suspendido. El proceso está en almacenamiento secundario pero está disponible para su ejecución tan pronto como sea cargado en memoria principal.





Las nuevas transiciones más importantes son las siguientes:

- Bloqueado > Bloqueado/Suspendido. Si no hay procesos listos, entonces al menos uno de los procesos bloqueados se transfiere al disco para hacer espacio para otro proceso que no se encuentra bloqueado. Esta transición puede realizarse incluso si hay procesos listos disponibles, si el sistema operativo determina que el proceso actualmente en ejecución o los procesos listos que desea ejecutar requieren más memoria principal para mantener un rendimiento adecuado.
- Bloqueado/Suspendido > Listo/Suspendido. Un proceso en el estado Bloqueado/Suspendido se mueve al estado Listo/Suspendido cuando sucede un evento al que estaba esperando. Nótese que esto requiere que la información de estado concerniente a un proceso suspendido sea accesible para el sistema operativo.
- Listo/Suspendido > Listo. Cuando no hay más procesos listos en memoria principal, el sistema operativo necesitará traer uno para continuar la ejecución. Adicionalmente, puede darse el caso de que un proceso en estado Listo/Suspendido tenga mayor prioridad que cualquiera de los procesos en estado Listo. En este caso, el sistema operativo puede haberse diseñado para determinar que es más importante traer un proceso de mayor prioridad que para minimizar el efecto del swapping.
- Listo > Listo/Suspendido. Normalmente, el sistema operativo preferirá suspender procesos bloqueados que un proceso Listo, porque un proceso Listo se puede ejecutar en ese momento, mientras que un proceso Bloqueado ocupa espacio de memoria y no se puede ejecutar. Sin embargo, puede ser necesario suspender un proceso Listo si con ello se consigue liberar un bloque suficientemente grande de memoria. También el sistema operativo puede decidir suspender un proceso Listo de baja prioridad antes que un proceso Bloqueado de alta prioridad, si se cree que el proceso Bloqueado estará pronto listo.
- Nuevo > Listo/Suspendido y Nuevo a Listo. Cuando se crea un proceso nuevo, puede añadirse a la cola de Listos o a la cola de Listos/Suspendidos. En cualquier caso, el sistema operativo puede crear un bloque de control de proceso (BCP) y reservar el espacio de direcciones del proceso. Puede ser preferible que el sistema operativo realice estas tareas internas cuanto antes, de forma que pueda disponer de suficiente cantidad de procesos no bloqueados. Sin embargo, con esta estrategia, puede ocurrir que no haya espacio suficiente en memoria principal para el nuevo proceso; de ahí el uso de la transición (Nuevo > Listo/Suspendido). Por otro lado, deseamos hacer hincapié en que la filosofía de creación de procesos diferida, haciéndolo cuanto más tarde, hace posible reducir la sobrecarga del sistema operativo y le permite realizar las tareas de creación de procesos cuando el sistema está lleno de procesos bloqueados.
- Bloqueado/Suspendido > Bloqueado. La incursión de esta transición puede parecer propia de un diseño más bien pobre. Después de todo, si hay un proceso que no está listo para ejecutar y que no está en memoria principal, ¿qué sentido tiene traerlo? Pero considérese el siguiente escenario: un proceso termina, liberando alguna memoria principal. Hay un proceso en la cola de Bloqueados/Suspendidos con mayor prioridad que todos los procesos en la cola de Listos/Suspendidos y el sistema operativo tiene motivos para creer que el evento que lo

bloquea va a ocurrir en breve. Bajo estas circunstancias, sería razonable traer el proceso Bloqueado a memoria por delante de los procesos Listos.

- Ejecutando > Listo/Suspendido. Normalmente, un proceso en ejecución se mueve a la cola de Listos cuando su tiempo de uso del procesador finaliza. Sin embargo, si el sistema operativo expulsa al proceso en ejecución porque un proceso de mayor prioridad en la cola de Bloqueado/Suspendido acaba de desbloquearse, el sistema operativo puede mover al proceso en ejecución directamente a la cola de Listos/Suspendidos y liberar parte de la memoria principal.
- De cualquier estado > Saliente. Habitualmente, un proceso termina cuando está ejecutando, bien porque ha completado su ejecución o debido a una condición de fallo. Sin embargo, en algunos sistemas operativos un proceso puede terminarse por el proceso que lo creó o cuando el proceso padre a su vez ha terminado. Si se permite esto, un proceso en cualquier estado puede moverse al estado Saliente.

Razones para suspender un proceso

Swapping	El sistema operativo necesita liberar suficiente memoria principal para traer un proceso en estado Listo de ejecución.
Otras razones del sistema operativo	El sistema operativo puede suspender un proceso en segundo plano o de utilidad o un proceso que se sospecha puede causar algún problema.
Solicitud interactiva del usuario	Un usuario puede desear suspender la ejecución de un programa con motivo de su depuración o porque está utilizando un recurso.
Temporización	Un proceso puede ejecutarse periódicamente (por ejemplo, un proceso monitor de estadísticas sobre el sistema) y puede suspenderse mientras espera el siguiente intervalo de ejecución.
Solicitud del proceso padre	Un proceso padre puede querer suspender la ejecución de un descendiente para examinar o modificar dicho proceso suspendido, o para coordinar la actividad de varios procesos descendientes.

3.3 Descripción de Procesos

El sistema operativo controla los eventos dentro del computador, planifica y activa los procesos para su ejecución por el procesador, reserva recursos para los mismos y responde a las solicitudes de servicios básicos de los procesos de usuario. Fundamentalmente, se piensa en el sistema operativo como en la entidad que gestiona el uso de recursos del sistema por parte de los procesos.

ESTRUCTURAS DE CONTROL DEL SO

El sistema operativo construye y mantiene tablas de información sobre cada entidad que gestiona.

Las **tablas de memoria** se usan para mantener un registro tanto de la memoria principal (real) como de la secundaria (virtual). Parte de la memoria principal está reservada para el uso del sistema operativo; el resto está disponible para el uso de los procesos. Los procesos se mantienen en memoria secundaria utilizando algún tipo de memoria virtual o técnicas de swapping.

El sistema operativo debe utilizar las **tablas de E/S** para gestionar los dispositivos de E/S y los canales del computador. Pero, en un instante determinado, un dispositivo E/S puede estar disponible o asignado a un proceso en particular. Si la operación de E/S se está realizando, el sistema operativo necesita conocer el estado de la operación y la dirección de memoria principal del área usada como fuente o destino de la transferencia de E/S.

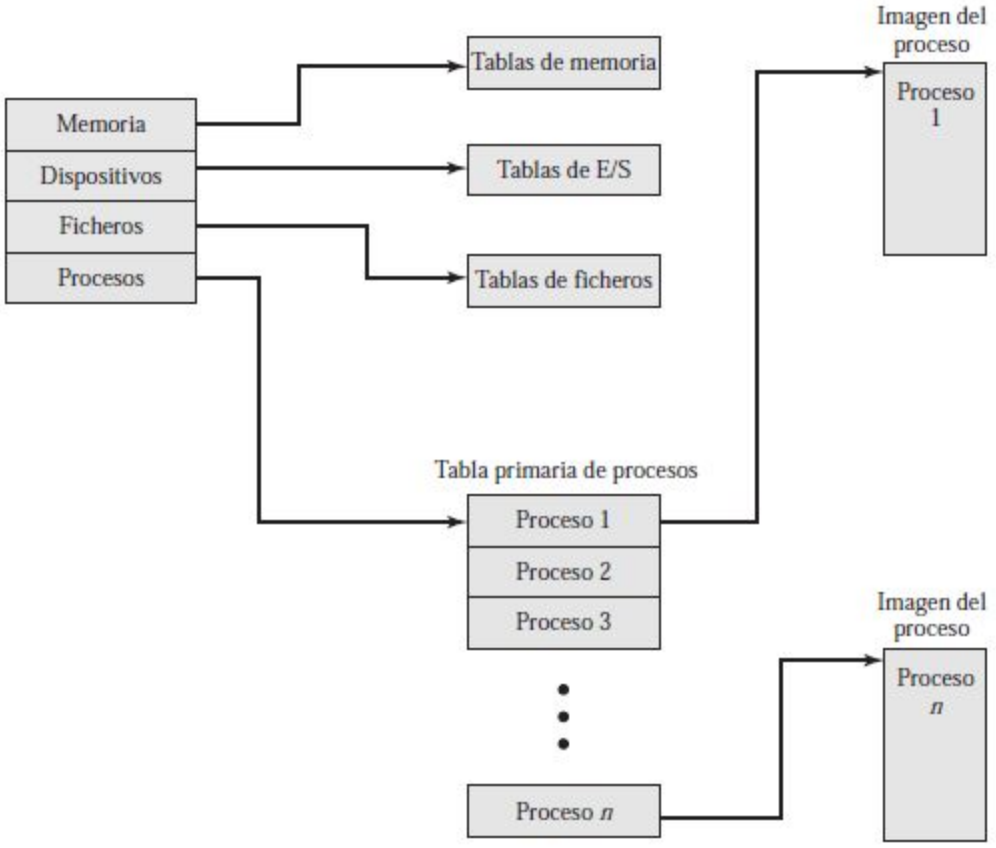
El sistema operativo también puede mantener las **tablas de ficheros**. Estas tablas proporcionan información sobre la existencia de ficheros, su posición en almacenamiento secundario, su estado actual, y otros atributos. La mayoría de, o prácticamente toda, esta información se puede gestionar por el sistema de ficheros, en cuyo caso el sistema operativo tiene poco o ningún conocimiento sobre los ficheros. En otros sistemas operativos, la gran mayoría del detalle de la gestión de ficheros sí que gestiona en el propio sistema operativo.

En último lugar, el sistema operativo debe mantener **tablas de procesos** para gestionar los procesos.

¿cómo puede el sistema operativo crear las tablas iniciales? Ciertamente, el sistema operativo debe tener alguna información sobre el entorno básico, tal como: cuánta memoria física existe, cuáles son los dispositivos de E/S y cuáles son sus identificadores, por ejemplo. Esto es una cuestión de configuración. Esto es, cuando el sistema operativo se inicializa, debe tener acceso a algunos datos de configuración que definen el entorno básico y estos datos se deben crear fuera del sistema operativo, con asistencia humana o por medio de algún software de autoconfiguración.

ESTRUCTURAS DE CONTROL DE PROCESOS

Se considerará qué información debe conocer el sistema operativo si quiere manejar y controlar los procesos. Primero, debe conocer dónde están localizados los procesos, y segundo, debe conocer los atributos de los procesos que quiere gestionar (por ejemplo, identificador de proceso y estado del mismo).



Como mínimo, un proceso debe incluir un programa o un conjunto de programas a ejecutar. Asociados con estos programas existen unas posiciones de memoria para los datos de variables locales y globales y de cualquier constante definida. Así, un proceso debe consistir en una cantidad suficiente de memoria para almacenar el programa y datos del mismo. Adicionalmente, la ejecución típica de un programa incluye una pila que se utiliza para registrar las llamadas a procedimientos y los parámetros pasados entre dichos procedimientos. Por último, cada proceso está asociado a un número de atributos que son utilizados por el sistema operativo para controlar el proceso. Normalmente, el conjunto de estos atributos se denomina bloque de control del proceso (BCP). Nos podemos referir al conjunto de programa, datos, pila, y atributos, como la **imagen del proceso**.

La posición de la imagen del proceso dependerá del esquema de gestión de memoria que se utilice. En el caso más simple, la imagen del proceso se mantiene como un bloque de memoria contiguo, o continuo. Este bloque se mantiene en memoria secundaria, habitualmente disco. Para que el sistema operativo pueda gestionar el proceso, al menos una pequeña porción de su imagen se debe mantener en memoria principal. Para ejecutar el proceso, la imagen del proceso completa se debe cargar en memoria principal o al menos en memoria virtual. Asimismo, el sistema operativo necesita conocer la posición en disco de cada proceso y, para cada proceso que se encuentre en memoria principal, su posición en dicha memoria.

Datos del usuario	La parte modificable del espacio de usuario. Puede incluir datos de programa, área de pila de usuario, y programas que puedan ser modificados.
Programa de usuario	El programa a ejecutar
Pila de sistema	Cada proceso tiene una o más pilas de sistema (LIFO) asociadas a él. Una pila se utiliza para almacenar los parámetros y las direcciones de retorno de los procedimientos y llamadas a sistema.
Bloque de control de proceso	Datos necesarios para que el sistema operativo pueda controlar los procesos (*véase* tabla 3.5).

Los sistemas operativos modernos suponen la existencia de un hardware de paginación que permite el uso de la memoria física no contigua, para dar soporte a procesos parcialmente residentes en la memoria principal. En esos casos, una parte de la imagen del proceso se puede encontrar en dicha memoria principal, con las restantes en memoria secundaria. De esta forma, las tablas mantenidas por sistema operativo deben mostrar la localización de cada página de la imagen del proceso.
Hay una tabla primaria de procesos con una entrada por cada proceso. Cada entrada contiene, al menos, un puntero a la imagen del proceso. Si la imagen del proceso contiene múltiples bloques, esta información se mantiene directamente en la tabla principal del proceso o bien está disponible por referencias cruzadas entre las entradas de las tablas de memoria. Por supuesto, esta es una representación genérica; un sistema operativo en particular puede tener su propia forma de organizar esta información de localización.

3.4 Control de procesos

MODOS DE EJECUCIÓN

El modo menos privilegiado a menudo se denomina **modo usuario**, porque los programas de usuario típicamente se ejecutan en este modo. El modo más privilegiado se denomina modo sistema, modo control o modo núcleo.

Se necesita proteger al sistema operativo y a las tablas clave del sistema, por ejemplo los bloques de control de proceso, de la interferencia con programas de usuario. En **modo núcleo**, el software tiene control completo del procesador y de sus instrucciones, registros, y memoria. Este nivel de control no es necesario y por seguridad no es recomendable para los programas de usuario.

En lo referente la primera cuestión, existe típicamente un bit en la palabra de estado de programa (PSW) que indica el modo de ejecución. Este bit se cambia como respuesta a determinados eventos. Habitualmente, cuando un usuario realiza una llamada a un servicio del sistema operativo o cuando una interrupción dispara la ejecución de una rutina del sistema operativo, este modo de ejecución se cambia a modo núcleo y; tras la finalización del servicio, el modo se fija de nuevo a modo usuario.

CREACIÓN DE PROCESOS (PASO A PASO)

Una vez que el sistema operativo decide, por cualquier motivo, crear un proceso procederá de la siguiente manera:

1. Asignar un identificador de proceso único al proceso. En este instante, se añade una nueva entrada a la tabla primaria de procesos, que contiene una entrada por proceso.
2. Reservar espacio para proceso. Esto incluye todos los elementos de la imagen del proceso. Para ello, el sistema operativo debe conocer cuánta memoria se requiere para el espacio de direcciones privado (programas y datos) y para la pila de usuario. Estos valores se pueden asignar por defecto basándonos en el tipo de proceso, o pueden fijarse en base a la solicitud de creación del trabajo remitido por el usuario. Si un proceso es creado por otro proceso, el proceso padre puede pasar los parámetros requeridos por el sistema operativo como parte de la solicitud de la creación de proceso. Si existe una parte del espacio direcciones compartido por este nuevo proceso, se fijan los enlaces apropiados. Por último, se debe reservar el espacio para el bloque de control de proceso (BCP).
3. Inicialización del bloque de control de proceso. La parte de identificación de proceso del BCP contiene el identificador del proceso así como otros posibles identificadores, tal como el indicador del proceso padre. En la información de estado de proceso del BCP, habitualmente se inicializa con la mayoría de entradas a 0, excepto el contador de programa (fijado en el punto entrada del programa) y los punteros de pila de sistema (fijados para definir los límites de la pila del proceso). La parte de información de control de procesos se inicializa en base a los valores por omisión, considerando también los atributos que han sido solicitados para este proceso. Por ejemplo, el estado del proceso se puede inicializar normalmente a Listo o Listo/Suspendido. La prioridad se puede fijar, por defecto, a la prioridad más baja, a menos que una solicitud explícita la eleve a una prioridad mayor. Inicialmente, el proceso no debe poseer ningún recurso (dispositivos de E/S, ficheros) a menos que exista una indicación explícita de ello o que haya sido heredados del padre.
4. Establecer los enlaces apropiados. Por ejemplo, si el sistema operativo mantiene cada cola del planificador como una lista enlazada, el nuevo proceso debe situarse en la cola de Listos o en la cola de Listos/Suspendidos.
5. Creación o expansión de otras estructuras de datos. Por ejemplo, el sistema operativo puede mantener un registro de auditoría por cada proceso que se puede utilizar posteriormente a efectos de facturación y/o de análisis de rendimiento del sistema.

CAMBIO DE PROCESO

Un cambio de proceso puede ocurrir en cualquier instante en el que el sistema operativo obtiene el control sobre el proceso actualmente en ejecución.

Primero, consideremos las **interrupciones** del sistema. Realmente, podemos distinguir, como hacen muchos sistemas, dos tipos diferentes de interrupciones de sistema, unas simplemente denominadas interrupciones, y las otras denominadas traps. Las primeras se producen por causa de algún tipo de evento que es externo e independiente al proceso actualmente en ejecución, por ejemplo la finalización de la operación de E/S. Las otras están asociadas a una condición de error o excepción

generada dentro del proceso que está ejecutando, como un intento de acceso no permitido a un fichero. Dentro de una interrupción ordinaria, el control se transfiere inicialmente al manejador de interrupción, que realiza determinadas tareas internas y que posteriormente salta a un rutina del sistema operativo, encargada de cada uno de los tipos de interrupciones en particular. Algunos ejemplos son:

- Interrupción de reloj. El sistema operativo determinar si el proceso en ejecución ha excedido o no la unidad máxima de tiempo de ejecución, denominada rodaja de tiempo (time slice).
- Interrupción de E/S. El sistema operativo determina qué acción de E/S ha ocurrido. Si la acción de E/S constituye un evento por el cual están esperando uno o más procesos, el sistema operativo mueve todos los procesos correspondientes al estado de Listos (y los procesos en estado Bloqueado/Suspendido al estado Listo/Suspendido). El sistema operativo puede decidir si reanuda la ejecución del proceso actualmente en estado Ejecutando o si lo expulsa para proceder con la ejecución de un proceso Listo de mayor prioridad.
- Fallo de memoria. El procesador se encuentra con una referencia a una dirección de memoria virtual, a una palabra que no se encuentra en memoria principal. El sistema operativo debe traer el bloque (página o segmento) que contiene la referencia desde memoria secundaria a memoria principal. Después de que se solicita la operación de E/S para traer el bloque a memoria, el proceso que causó el fallo de memoria se pasa al estado de Bloqueado; el sistema operativo realiza un cambio de proceso y pone a ejecutar a otro proceso. Después de que el bloque de memoria solicitado se haya traído, el proceso pasará al estado Listo.

Con un trap, el sistema operativo conoce si una condición de error o de excepción es irreversible. Si es así, el proceso en ejecución se pone en el estado Saliente y se hace un cambio de proceso. De no ser así, el sistema operativo actuará dependiendo de la naturaleza del error y su propio diseño. Se puede intentar un procedimiento de recuperación o simplemente la notificación al usuario, pudiendo implicar tanto un cambio de proceso como la continuación de la ejecución del proceso actual.

Por último, el sistema operativo se puede activar por medio de una llamada al sistema procedente del programa en ejecución. Por ejemplo, si se está ejecutando un proceso y se ejecuta una operación que implica una llamada de E/S, como abrir un archivo. Esta llamada implica un salto a una rutina que es parte del código del sistema operativo. La realización de una llamada al sistema puede implicar en algunos casos que el proceso que la realiza pase al estado de Bloqueado.

CAMBIO DE MODO

Recordando esto, en la fase de interrupción, el procesador comprueba que no exista ninguna interrupción pendiente, indicada por la presencia de una señal de interrupción. Si no hay interrupciones pendientes, el procesador pasa a la fase de búsqueda de instrucción, siguiendo con el programa del proceso actual. Si hay una interrupción pendiente, el proceso actúa de la siguiente manera:

1. Coloca el contador de programa en la dirección de comienzo de la rutina del programa manejador de la interrupción.
2. Cambia de modo usuario a modo núcleo de forma que el código de tratamiento de la interrupción pueda incluir instrucciones privilegiadas.

El procesador, acto seguido, pasa a la fase de búsqueda de instrucción y busca la primera instrucción del programa de manejo de interrupción, que dará servicio a la misma. En este punto, habitualmente, el contexto del proceso que se ha interrumpido se salvaguarda en el bloque de control de proceso del programa interrumpido.

Una pregunta que se puede plantear es, ¿qué constituye el contexto que se debe salvaguardar? La respuesta es que se trata de toda la información que se puede ver alterada por la ejecución de la rutina de interrupción y que se necesitará para la continuación del proceso que ha sido interrumpido. De esta forma, se debe guardar la parte del bloque de control del proceso que hace referencia a la información de estado del procesador. Esto incluye el contador de programa, otros registros del procesador, y la información de la pila.

¿Se necesita hacer algo más? Eso depende de qué ocurra luego. El manejador de interrupción es habitualmente un pequeño programa que realiza unas pocas tareas básicas relativas a la interrupción. Por ejemplo, borra el flag o indicador que señala la presencia de interrupciones. Puede enviar una confirmación a la entidad que lanzó dicha interrupción, como por ejemplo el módulo de E/S. Y puede realizar algunas tareas internas variadas relativas a los efectos del evento que causó la interrupción. Por ejemplo, si la interrupción se refiere a un evento de E/S, el manejador de interrupción comprobará la existencia o no de una condición de error. Si ha ocurrido un error, el manejador mandará una señal al proceso que solicitó dicha operación de E/S. Si la interrupción proviene del reloj, el manejador la va a pasar el control al activador, el cual decidirá pasar a otro proceso debido a que la rodaja de tiempo asignada a ese proceso ha expirado.

¿Qué pasa con el resto de información del bloque de control de proceso? Si a esta interrupción le sigue un cambio de proceso a otro proceso, se necesita hacer algunas cosas más. Sin embargo, en muchos sistemas operativos, la existencia de una interrupción no implica necesariamente un cambio de proceso. Es posible, por tanto, que después de la ejecución de la rutina de interrupción, la ejecución se reanude con el mismo proceso. En esos casos sólo se necesita salvaguardar la información del estado del procesador cuando se produce la interrupción y restablecerlo cuando se reanude la ejecución del proceso. Habitualmente, las operaciones de salvaguarda y recuperación se realizan por hardware.

CAMBIO DEL ESTADO DEL PROCESO

Está claro, por tanto, que el cambio de modo es un concepto diferente del cambio de proceso. Un cambio de modo puede ocurrir sin que se cambie el estado del proceso actualmente en estado Ejecutando. En dicho caso, la salvaguarda del estado y su posterior restauración comportan sólo una ligera sobrecarga. Sin embargo, si el proceso actualmente en estado Ejecutando, se va a mover a

cualquier otro estado (Listo, Bloqueado, etc.), entonces el sistema operativo debe realizar cambios sustanciales en su entorno. Los pasos que se realizan para un cambio de proceso completo son:

1. Salvar el estado del procesador, incluyendo el contador de programa y otros registros.
2. Actualizar el bloque de control del proceso que está actualmente en el estado Ejecutando. Esto incluye cambiar el estado del proceso a uno de los otros estados (Listo, Bloqueado, Listo/ Suspendido, o Saliente). También se tienen que actualizar otros campos importantes, incluyendo la razón por la cual el proceso ha dejado el estado de Ejecutando y demás información de auditoría.
3. Mover el bloque de control de proceso a la cola apropiada (Listo, Bloqueado en el evento i, Listo/Suspendido).
4. Selección de un nuevo proceso a ejecutar; esta cuestión se analiza con más detalle en la Parte Cuatro.
5. Actualizar el bloque de control del proceso elegido. Esto incluye pasarlo al estado Ejecutando.
6. Actualizar las estructuras de datos de gestión de memoria. Esto se puede necesitar, dependiendo de cómo se haga la traducción de direcciones; estos aspectos se cubrirán en la Parte Tres.
7. Restaurar el estado del procesador al que tenía en el momento en el que el proceso seleccionado salió del estado Ejecutando por última vez, leyendo los valores anteriores de contador de programa y registros.

Por tanto, el cambio de proceso, que implica un cambio en el estado, requiere un mayor esfuerzo que un cambio de modo.

4. Hilos, SMP y micronucleos

4.1 Procesos e Hilos

Se ha presentado el concepto de un proceso como poseedor de dos características:

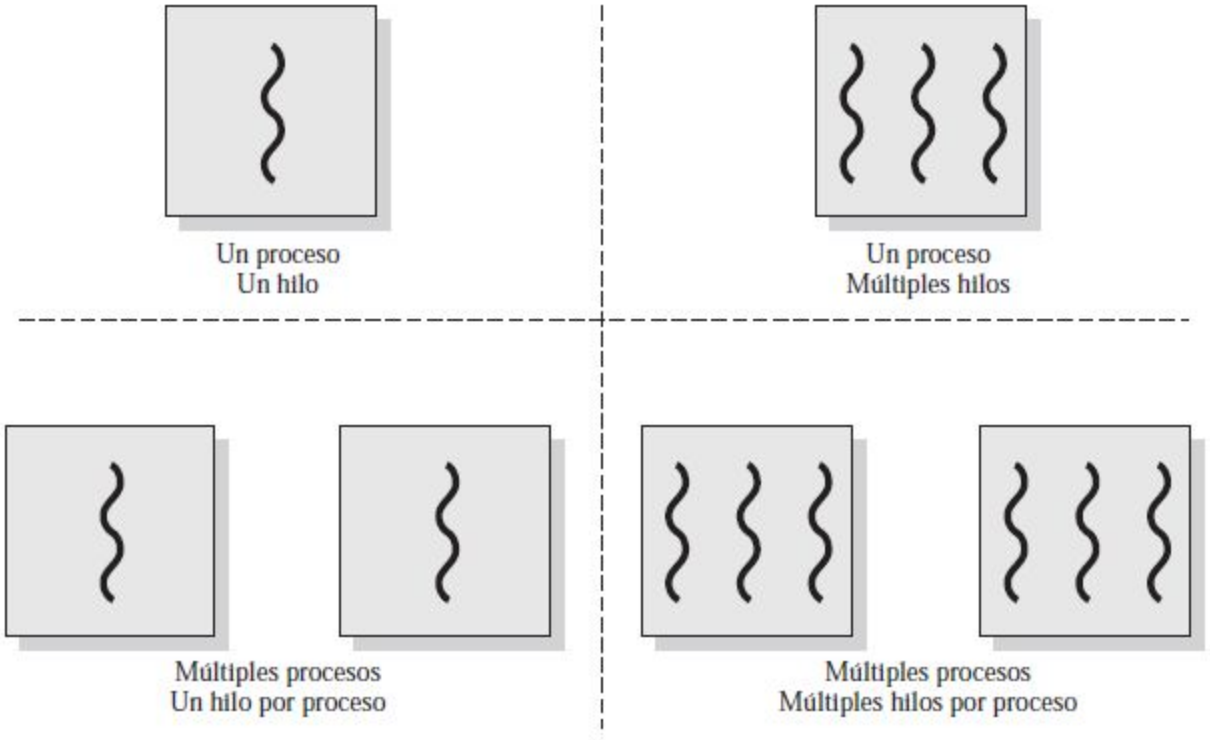
- **Propiedad de recursos**. Un proceso incluye un espacio de direcciones virtuales para el manejo de la imagen del proceso; como ya se explicó en el Capítulo 3 la imagen de un proceso es la colección de programa, datos, pila y atributos definidos en el bloque de control del proceso. De vez en cuando a un proceso se le puede asignar control o propiedad de recursos tales como la memoria principal, canales E/S, dispositivos E/S y archivos.
- **Planificación/ejecución.** La ejecución de un proceso sigue una ruta de ejecución (traza) a través de uno o más programas.

En la mayor parte de los sistemas operativos tradicionales, estas dos características son, realmente, la esencia de un proceso. Sin embargo, debe quedar muy claro que estas dos características son

independientes y podrían ser tratadas como tales por el sistema operativo. Así se hace en diversos sistemas operativos, sobre todo en los desarrollados recientemente. Para distinguir estas dos características, la unidad que se activa se suele denominar hilo (thread), o proceso ligero, mientras que la unidad de propiedad de recursos se suele denominar proceso o tarea1.

MULTIHILO

Multihilo se refiere a la capacidad de un sistema operativo de dar soporte a múltiples hilos de ejecución en un solo proceso. El enfoque tradicional de un solo hilo de ejecución por proceso, en el que no se identifica con el concepto de hilo, se conoce como estrategia monohilo.



En un entorno multihilo, un proceso se define como la unidad de asignación de recursos y una unidad de protección. Se asocian con procesos los siguientes:

- Un espacio de direcciones virtuales que soporta la imagen del proceso.
- Acceso protegido a procesadores, otros procesos (para comunicación entre procesos), archivos y recursos de E/S (dispositivos y canales).

Dentro de un proceso puede haber uno o más hilos, cada uno con:

- Un estado de ejecución por hilo (Ejecutando, Listo, etc.).

- Un contexto de hilo que se almacena cuando no está en ejecución; una forma de ver a un hilo es como un contador de programa independiente dentro de un proceso.
- Una pila de ejecución.
- Por cada hilo, espacio de almacenamiento para variables locales.
- Acceso a la memoria y recursos de su proceso, compartido con todos los hilos de su mismo proceso.

En un modelo de proceso monohilo (es decir, no existe el concepto de hilo), la representación de un proceso incluye su bloque de control de proceso y el espacio de direcciones de usuario, además de las pilas de usuario y núcleo para gestionar el comportamiento de las llamadas/retornos en la ejecución de los procesos. Mientras el proceso está ejecutando, los registros del procesador se controlan por ese proceso y, cuando el proceso no se está ejecutando, se almacena el contenido de estos registros. En un entorno multihilo, sigue habiendo un único bloque de control del proceso y un espacio de direcciones de usuario asociado al proceso, pero ahora hay varias pilas separadas para cada hilo, así como un bloque de control para cada hilo que contiene los valores de los registros, la prioridad, y otra información relativa al estado del hilo.

De esta forma, todos los hilos de un proceso comparten el estado y los recursos de ese proceso, residen en el mismo espacio de direcciones y tienen acceso a los mismos datos. Cuando un hilo cambia determinados datos en memoria, otros hilos ven los resultados cuando acceden a estos datos. Si un hilo abre un archivo con permisos de lectura, los demás hilos del mismo proceso pueden también leer ese archivo.

Alguna de las ventajas:

- Lleva mucho menos tiempo crear un nuevo hilo en un proceso existente que crear un proceso totalmente nuevo.
- Lleva menos tiempo finalizar un hilo que un proceso.
- Lleva menos tiempo cambiar entre dos hilos dentro del mismo proceso.
- Los hilos mejoran la eficiencia de la comunicación entre diferentes programas que están ejecutando. En la mayor parte de los sistemas operativos, la comunicación entre procesos independientes requiere la intervención del núcleo para proporcionar protección y los mecanismos necesarios de comunicación. Sin embargo, ya que los hilos dentro de un mismo proceso comparten memoria y archivos, se pueden comunicar entre ellos sin necesidad de invocar al núcleo.

De esta forma, si se desea implementar una aplicación o función como un conjunto de unidades de ejecución relacionadas, es mucho más eficiente hacerlo con un conjunto de hilos que con un conjunto de procesos independientes.

A veces los hilos son también útiles en un solo procesador ya que ayudan a simplificar la estructura de programas que realizan varias funciones diferentes.

Suspender un proceso implica expulsar el espacio de direcciones de un proceso de memoria principal para dejar hueco a otro espacio de direcciones de otro proceso. Ya que todos los hilos de un proceso comparten el mismo espacio de direcciones, todos los hilos se suspenden al mismo tiempo. De forma similar, la finalización de un proceso finaliza todos los hilos de ese proceso.

FUNCIONALIDADES DE LOS HILOS

Los hilos, al igual que los procesos, tienen estados de ejecución y se pueden sincronizar entre ellos

Estados de los hilos. Igual que con los procesos, los principales estados de los hilos son: Ejecutando, Listo y Bloqueado.

Hay cuatro operaciones básicas relacionadas con los hilos que están asociadas con un cambio de estado del hilo:

- Creación. Cuando se crea un nuevo proceso, también se crea un hilo de dicho proceso. Posteriormente, un hilo del proceso puede crear otro hilo dentro del mismo proceso, proporcionando un puntero a las instrucciones y los argumentos para el nuevo hilo. Al nuevo hilo se le proporciona su propio registro de contexto y espacio de pila y se coloca en la cola de Listos.
- Bloqueo. Cuando un hilo necesita esperar por un evento se bloquea, almacenando los registros de usuario, contador de programa y punteros de pila. El procesador puede pasar a ejecutar otro hilo en estado Listo, dentro del mismo proceso o en otro diferente.
- Desbloqueo. Cuando sucede el evento por el que el hilo está bloqueado, el hilo se pasa a la cola de Listos.
- Finalización. Cuando se completa un hilo, se liberan su registro de contexto y pilas.

Sincronización de hilos

Todos los hilos de un proceso comparten el mismo espacio de direcciones y otros recursos, como por ejemplo, los archivos abiertos. Cualquier alteración de un recurso por cualquiera de los hilos, afecta al entorno del resto de los hilos del mismo proceso. Por tanto, es necesario sincronizar las actividades de los hilos para que no interfieran entre ellos o corrompan estructuras de datos.

HILOS A NIVEL DE USUARIO Y A NIVEL DE NUCLEO

Existen dos amplias categorías de implementación de hilos: hilos de nivel de usuario (user-level threads, ULT) e hilos de nivel de núcleo (kernel-level threads, KLT).

Hilos de nivel de usuario. En un entorno ULT puro, la aplicación gestiona todo el trabajo de los hilos y el núcleo no es consciente de la existencia de los mismos.

Cualquier aplicación puede programarse para ser multihilo a través del uso de una biblioteca de hilos, que es un paquete de rutinas para la gestión de ULT. La biblioteca de hilos contiene código

para la creación y destrucción de hilos, para paso de mensajes y datos entre los hilos, para planificar la ejecución de los hilos, y para guardar y restaurar el contexto de los hilos.

El núcleo no es consciente de esta actividad. El núcleo continúa planificando el proceso como una unidad y asigna al proceso un único estado.

Hay que advertir que un proceso puede interrumpirse, bien por finalizar su porción de tiempo o bien por ser expulsado por un proceso de mayor prioridad, mientras está ejecutando código de la biblioteca de los hilos. De esta forma, un proceso puede estar en medio de una transición de un hilo a otro hilo cuando se interrumpe. Cuando se reanuda el proceso, la ejecución continúa con la biblioteca de los hilos, que completa el cambio de hilo y pasa el control al nuevo hilo del proceso.

El uso de ULT en lugar de KLT, presenta las siguientes ventajas:

- El cambio de hilo no requiere privilegios de modo núcleo porque todas las estructuras de datos de gestión de hilos están en el espacio de direcciones de usuario de un solo proceso. Por consiguiente, el proceso no cambia a modo núcleo para realizar la gestión de hilos. Esto ahorra la sobrecarga de dos cambios de modo (usuario a núcleo; núcleo a usuario).
- La planificación puede especificarse por parte de la aplicación.
- Los ULT pueden ejecutar en cualquier sistema operativo.

Hay dos desventajas de los ULT en comparación con los KLT:

- En un sistema operativo típico muchas llamadas al sistema son bloqueantes. Como resultado, cuando un ULT realiza una llamada al sistema, no sólo se bloquea ese hilo, sino que se bloquean todos los hilos del proceso.
- En una estrategia pura ULT, una aplicación multihilo no puede sacar ventaja del multiproceso. El núcleo asigna el proceso a un solo procesador al mismo tiempo.

Hay formas de afrontar estos dos problemas. Por ejemplo, ambos problemas pueden superarse escribiendo una aplicación de múltiples procesos en lugar de múltiples hilos. Pero este enfoque elimina la principal ventaja de los hilos: cada cambio es un cambio de proceso en lugar de un cambio de hilo, lo que genera una gran sobrecarga. Otra forma de solucionar el problema de hilos que se bloquean es una técnica denominada jacketing (revestimiento). El objetivo de esta técnica es convertir una llamada al sistema bloqueante en una llamada al sistema no bloqueante.

Hilos a nivel de núcleo. En un entorno KLT puro, el núcleo gestiona todo el trabajo de gestión de hilos. No hay código de gestión de hilos en la aplicación, solamente una interfaz de programación de aplicación (API) para acceder a las utilidades de hilos del núcleo. Windows es un ejemplo de este enfoque.

Cualquier aplicación puede programarse para ser multihilo. Todos los hilos de una aplicación se mantienen en un solo proceso. El núcleo mantiene información de contexto del proceso como una

entidad y de los hilos individuales del proceso. La planificación realizada por el núcleo se realiza a nivel de hilo. Este enfoque resuelve los dos principales inconvenientes del enfoque ULT. Primero, el núcleo puede planificar simultáneamente múltiples hilos de un solo proceso en múltiples procesadores. Segundo, si se bloquea un hilo de un proceso, el núcleo puede planificar otro hilo del mismo proceso. Otra ventaja del enfoque KLT es que las rutinas del núcleo pueden ser en sí mismas multihilo.

La principal desventaja del enfoque KLT en comparación con el enfoque ULT es que la transferencia de control de un hilo a otro del mismo proceso requiere un cambio de modo al núcleo.

Enfoques combinados. Algunos sistemas operativos proporcionan utilidades combinadas ULT/KLT. Solaris es el principal ejemplo de esto. En un sistema combinado, la creación de hilos se realiza por completo en el espacio de usuario, como la mayor parte de la planificación y sincronización de hilos dentro de una aplicación. Los múltiples ULT de una aplicación se asocian en un número (menor o igual) de KLT. El programador debe ajustar el número de KLT para una máquina y aplicación en particular para lograr los mejores resultados posibles.
En los enfoques combinados, múltiples hilos de la misma aplicación pueden ejecutar en paralelo en múltiples procesadores, y una llamada al sistema bloqueante no necesita bloquear el proceso completo. Si el sistema está bien diseñado, este enfoque debería combinar las ventajas de los enfoques puros ULT y KLT, minimizando las desventajas.

MODELOS COMBINADOS

Hilos:Procesos	Descripción	Sistemas de Ejemplo
1:1	Cada hilo de ejecución es un único proceso con su propio espacio de direcciones y recursos.	Implementaciones UNIX tradicionales.
M:1	Un proceso define un espacio de direcciones y pertenencia dinámica de recursos. Se pueden crear y ejecutar múltiples hilos en ese proceso.	Windows NT, Solaris, Linux, OS/2, OS/390, MACH.
1:M	Un hilo puede migrar de un entorno de proceso a otro. Esto permite a los hilos moverse fácilmente entre distintos sistemas.	Ra (Clouds), Emerald.
M:N	Combina atributos de M:1 y casos 1:M.	TRIX.

5 Concurrencia. Exclusión mutua y sincronización

Los temas centrales del diseño de sistemas operativos están todos relacionados con la gestión de procesos e hilos:

- Multiprogramación. Gestión de múltiples procesos dentro de un sistema monoprocesador.
- Multiprocesamiento. Gestión de múltiples procesos dentro de un multiprocesador.
- Procesamiento distribuido. Gestión de múltiples procesos que ejecutan sobre múltiples sistemas de cómputo distribuidos.

La concurrencia es fundamental en todas estas áreas y en el diseño del sistema operativo.

La concurrencia aparece en tres contextos diferentes:

- Múltiples aplicaciones. La multiprogramación fue ideada para permitir compartir dinámicamente el tiempo de procesamiento entre varias aplicaciones activas.
- Aplicaciones estructuradas. Como extensión de los principios del diseño modular y de la programación estructurada, algunas aplicaciones pueden ser programadas eficazmente como un conjunto de procesos concurrentes.
- Estructura del sistema operativo. Las mismas ventajas constructivas son aplicables a la programación de sistemas y, de hecho, los sistemas operativos son a menudo implementados en sí mismos como un conjunto de procesos o hilos.

TERMINOS CLAVES

sección crítica (*critical section*)	Sección de código dentro de un proceso que requiere acceso a recursos compartidos y que no puede ser ejecutada mientras otro proceso esté en una sección de código correspondiente.
interbloqueo (*deadlock*)	Situación en la cual dos o más procesos son incapaces de actuar porque cada uno está esperando que alguno de los otros haga algo.
círculo vicioso (*livelock*)	Situación en la cual dos o más procesos cambian continuamente su estado en respuesta a cambios en los otros procesos, sin realizar ningún trabajo útil.
exclusión mutua (*mutual exclusion*)	Requisito de que cuando un proceso esté en una sección crítica que acceda a recursos compartidos, ningún otro proceso pueda estar en una sección crítica que acceda a ninguno de esos recursos compartidos.
condición de carrera (*race condition*)	Situación en la cual múltiples hilos o procesos leen y escriben un dato compartido y el resultado final depende de la coordinación relativa de sus ejecuciones.
inanición (*starvation*)	Situación en la cual un proceso preparado para avanzar es soslayado indefinidamente por el planificador; aunque es capaz de avanzar, nunca se le escoge.

5.1 Principios de la concurrencia

En un sistema multiprogramado de procesador único, los procesos se entrelazan en el tiempo para ofrecer la apariencia de ejecución simultánea.
Se plantean las siguientes dificultades:

- La compartición de recursos globales está cargada de peligros. Por ejemplo, si dos procesos utilizan ambos la misma variable global y ambos realizan lecturas y escrituras sobre esa variable, entonces el orden en que se ejecuten las lecturas y escrituras es crítico.
- Para el sistema operativo es complicado gestionar la asignación de recursos de manera óptima. Por ejemplo, el proceso A puede solicitar el uso de un canal concreto de E/S, y serle concedido el control, y luego ser suspendido justo antes de utilizar ese canal. Puede no ser deseable que el sistema operativo simplemente bloquee el canal e impida su utilización por otros procesos; de hecho esto puede conducir a una condición de interbloqueo.
- Llega a ser muy complicado localizar errores de programación porque los resultados son típicamente no deterministas y no reproducibles.

Todas las dificultades precedentes se presentan también en un sistema multiprocesador, porque aquí tampoco es predecible la velocidad relativa de ejecución de los procesos.

CONDICIÓN DE CARRERA
Una condición de carrera sucede cuando múltiples procesos o hilos leen y escriben datos de manera que el resultado final depende del orden de ejecución de las instrucciones en los múltiples procesos.

Como primer ejemplo, suponga que dos procesos, P1 y P2, comparten la variable global a. En algún punto de su ejecución, P1 actualiza a al valor 1 y, en el mismo punto en su ejecución, P2 actualiza a al valor 2. Así, las dos tareas compiten en una carrera por escribir la variable a. En este ejemplo el «perdedor» de la carrera (el proceso que actualiza el último) determina el valor de a.

INTERACCIÓN DE PROCESOS
Podemos clasificar las formas en que los procesos interaccionan en base al grado en que perciben la existencia de cada uno de los otros. Se enumera tres posibles grados de percepción más las consecuencias de cada uno:

- Procesos que no se perciben entre sí. Son procesos independientes que no se pretende que trabajen juntos. El mejor ejemplo de esta situación es la multiprogramación de múltiples procesos independientes. Estos bien pueden ser trabajos por lotes o bien sesiones interactivas o una mezcla. Aunque los procesos no estén trabajando juntos, el sistema operativo necesita preocuparse de la competencia por recursos. Por ejemplo, dos aplicaciones independientes pueden querer ambas acceder al mismo disco, fichero o impresora. El sistema operativo debe regular estos accesos.

- Procesos que se perciben indirectamente entre sí. Son procesos que no están necesariamente al tanto de la presencia de los demás mediante sus respectivos ID de proceso, pero que comparten accesos a algún objeto, como un buffer de E/S. Tales procesos exhiben cooperación en la compartición del objeto común.
- Procesos que se perciben directamente entre sí. Son procesos capaces de comunicarse entre sí vía el ID del proceso y que son diseñados para trabajar conjuntamente en cierta actividad. De nuevo, tales procesos exhiben cooperación.

Competencia entre procesos por recursos. Los procesos concurrentes entran en conflicto entre ellos cuando compiten por el uso del mismo recurso. En su forma pura, puede describirse la situación como sigue. Dos o más procesos necesitan acceso a un recurso durante el curso de su ejecución. Ningún proceso se percata de la existencia de los otros procesos y ninguno debe verse afectado por la ejecución de los otros procesos. Esto conlleva que cada proceso debe dejar inalterado el estado de cada recurso que utilice. Ejemplos de recursos son los dispositivos de E/S, la memoria, el tiempo de procesador y el reloj.

No hay intercambio de información entre los procesos en competencia. No obstante, la ejecución de un proceso puede afectar al comportamiento de los procesos en competencia. En concreto, si dos procesos desean ambos acceder al mismo recurso único, entonces, el sistema operativo reservará el recurso para uno de ellos, y el otro tendrá que esperar. Por tanto, el proceso al que se le deniega el acceso será ralentizado. En un caso extremo, el proceso bloqueado puede no conseguir nunca el recurso y por tanto no terminar nunca satisfactoriamente.

En el caso de procesos en competencia, deben afrontarse tres problemas de control. Primero está la necesidad de exclusión mutua. Supóngase que dos o más procesos requieren acceso a un recurso único no compartible, como una impresora. Durante el curso de la ejecución, cada proceso estará enviando mandatos al dispositivo de E/S, recibiendo información de estado, enviando datos o recibiendo datos. Nos referiremos a tal recurso como un recurso crítico, y a la porción del programa que lo utiliza como la sección crítica del programa. Es importante que sólo se permita un programa al tiempo en su sección crítica. No podemos simplemente delegar en el sistema operativo para que entienda y aplique esta restricción porque los detalles de los requisitos pueden no ser obvios. En el caso de una impresora, por ejemplo, queremos que cualquier proceso individual tenga el control de la impresora mientras imprime el fichero completo. De otro modo, las líneas de los procesos en competencia se intercalarían.

La aplicación de la exclusión mutua crea dos problemas de control adicionales. Uno es el del **interbloqueo**.
Cada proceso está esperando por uno de los dos recursos. Ninguno liberará el recurso que ya posee hasta haber conseguido el otro recurso y realizado la función que requiere ambos recursos. Los dos procesos están interbloqueados. Un último problema de control es la inanición.

Puede ocurrir el caso de denegársele indefinidamente el acceso al recurso, aunque no suceda un interbloqueo.
El control de la competencia involucra inevitablemente al sistema operativo ya que es quien ubica los recursos. Además, los procesos necesitarán ser capaces por sí mismos de expresar de alguna manera el requisito de exclusión mutua, como bloqueando el recurso antes de usarlo. Cualquier solución involucrará algún apoyo del sistema operativo, como proporcionar un servicio de bloqueo.

Cooperación entre procesos vía compartición. El caso de cooperación vía compartición cubre procesos que interaccionan con otros procesos sin tener conocimiento explícito de ellos. Por ejemplo, múltiples procesos pueden tener acceso a variables compartidas o a ficheros o bases de datos compartidas. Los procesos pueden usar y actualizar los datos compartidos sin referenciar otros procesos pero saben que otros procesos pueden tener acceso a los mismos datos. Así, los procesos deben cooperar para asegurar que los datos que comparten son manipulados adecuadamente. Los mecanismos de control deben asegurar la integridad de los datos compartidos.

Dado que los datos están contenidos en recursos (dispositivos, memoria), los problemas de control de exclusión mutua, interbloqueo e inanición están presentes de nuevo. La única diferencia es que los datos individuales pueden ser accedidos de dos maneras diferentes, lectura y escritura, y sólo las operaciones de escritura deben ser mutuamente exclusivas.

Sin embargo, por encima de estos problemas, surge un nuevo requisito: el de la **coherencia de datos**. Como ejemplo sencillo, considérese una aplicación de contabilidad en la que pueden ser actualizados varios datos individuales. Supóngase que dos datos individuales a y b han de ser mantenidos en la relación a = b. Esto es, cualquier programa que actualice un valor, debe también actualizar el otro para mantener la relación.
Así se ve que el concepto de sección crítica es importante en el caso de cooperación por compartición. Las mismas funciones abstractas *entrarcritica* y *salircritica* tratadas anteriormente pueden usarse aquí. En este caso, el argumento de las funciones podría ser una variable, un fichero o cualquier otro objeto compartido. Es más, si las secciones críticas se utilizan para conseguir integridad de datos, entonces puede no haber recurso o variable concreta que pueda ser identificada como argumento. En este caso, puede pensarse en el argumento como un identificador compartido entre los procesos concurrentes que identifica las secciones críticas que deben ser mutuamente exclusivas.

Cooperación entre procesos vía comunicación. Cuando los procesos cooperan vía comunicación, en cambio, los diversos procesos involucrados participan en un esfuerzo común que los vincula a todos ellos. La comunicación proporciona una manera de sincronizar o coordinar actividades varias. Típicamente, la comunicación se fundamenta en mensajes de algún tipo.
Dado que en el acto de pasar mensajes los procesos no comparten nada, la exclusión mutua no es un requisito de control en este tipo de cooperación. Sin embargo, los problemas de interbloqueo e inanición están presentes

REQUISITOS PARA EXCLUSION MUTUA

- La exclusión mutua debe hacerse cumplir: sólo se permite un proceso al tiempo dentro de su sección crítica, de entre todos los procesos que tienen secciones críticas para el mismo recurso u objeto compartido.
- Un proceso que se pare en su sección no crítica debe hacerlo sin interferir con otros procesos.
- No debe ser posible que un proceso que solicite acceso a una sección crítica sea postergado indefinidamente: ni interbloqueo ni inanición.
- Cuando ningún proceso esté en una sección crítica, a cualquier proceso que solicite entrar en su sección crítica debe permitirsele entrar sin demora.
- No se hacen suposiciones sobre las velocidades relativas de los procesos ni sobre el número de procesadores.
- Un proceso permanece dentro de su sección crítica sólo por un tiempo finito.

Hay varias maneras de satisfacer los requisitos para la exclusión mutua. Una manera es delegar la responsabilidad en los procesos que desean ejecutar concurrentemente. Esos procesos, ya sean programas del sistema o programas de aplicación, estarían obligados a coordinarse entre sí para cumplir la exclusión mutua, sin apoyo del lenguaje de programación ni del sistema operativo. Podemos referirnos a esto como soluciones software. Aunque este enfoque es propenso a una alta sobrecarga de procesamiento y a errores, sin duda es útil examinar estas propuestas para obtener una mejor comprensión de la complejidad de la programación concurrente.Un segundo enfoque es proporcionar cierto nivel de soporte dentro del sistema operativo o del lenguaje de programación.

5.2 Exclusion Mutua: Soporte Hardware

DESHABILITAR INTERRUPCIONES

En una máquina monoprocesador, los procesos concurrentes no pueden solaparse, sólo pueden entrelazarse. Es más, un proceso continuará ejecutando hasta que invoque un servicio del sistema operativo o hasta que sea interrumpido. Por tanto, para garantizar la exclusión mutua, basta con impedir que un proceso sea interrumpido. Esta técnica puede proporcionarse en forma de primitivas definidas por el núcleo del sistema para deshabilitar y habilitar las interrupciones.

Dado que la sección crítica no puede ser interrumpida, se garantiza la exclusión mutua. El precio de esta solución, no obstante, es alto. La eficiencia de ejecución podría degradarse notablemente porque se limita la capacidad del procesador de entrelazar programas. Un segundo problema es que esta solución no funcionará sobre una arquitectura multiprocesador. Cuando el sistema de cómputo incluye más de un procesador, es posible (y típico) que se estén ejecutando al tiempo más de un proceso. En este caso, deshabilitar interrupciones no garantiza exclusión mutua.

INSTRUCCIONES MÁQUINA ESPECIALES

En una configuración multiprocesador, varios procesadores comparten acceso a una memoria principal común.

A un nivel hardware, como se mencionó, el acceso a una posición de memoria excluye cualquier otro acceso a la misma posición. Con este fundamento, los diseñadores de procesadores han propuesto varias instrucciones máquina que llevan a cabo dos acciones atómicamente, como leer y escribir o leer y comprobar, sobre una única posición de memoria con un único ciclo de búsqueda de instrucción. Durante la ejecución de la instrucción, el acceso a la posición de memoria se le bloquea a toda otra instrucción que referencie esa posición. Típicamente, estas acciones se realizan en un único ciclo de instrucción.

Como desventajas tenemos que se utiliza espera activa, hay riesgo de inanición y de interbloqueo.

5.3 Semaforos

El principio fundamental es éste: dos o más procesos pueden cooperar por medio de simples señales, tales que un proceso pueda ser obligado a parar en un lugar específico hasta que haya recibido una señal específica.

Un semáforo S es una variable entera a la que, dejando aparte la inicialización, sólo se accede mediante dos operaciones atómicas estándar: wait y signal.

Wait (S) {
While S <= 0
; //none
S--;
}
	Signal (S) {
	S ++;
	}

Los semáforos se diferencian entre semáforos contadores y binarios o mutex. El valor de un semáforo contador puede variar en un dominio no restringido, mientras que el valor de un semáforo binario sólo puede ser 0 o 1.

Los semáforos contadores se pueden usar para controlar el acceso a un determinado recurso formado por un número finito de instancias. El semáforo se inicializa con el número de recursos disponibles. Cada proceso que desee acceder a un recurso ejecuta una operación wait() en el semáforo decrementando la cuenta. Cuando la cuenta del semáforo llega a cero, todos los recursos estarán en uso. Después, los procesos que deseen usar el recurso se bloquearán hasta que la cuenta sea mayor a cero.

La principal desventaja de los semáforos es que requiere una espera activa. Mientras un proceso está en su sección crítica, cualquier otro proceso que intente entrar en su sección crítica deberá ejecutar continuamente un bucle en el código de entrada verificando si puede acceder o no. Para solucionar esto podemos modificar las operaciones atómicas sobre el semáforo. Cuando un proceso ejecuta un wait y determina que el valor del semáforo es no positivo, tiene que esperar. Sin embargo en lugar de entrar en una espera activa, el proceso puede bloquearse a sí mismo y colocarse en en una cola de espera asociada al semáforo. Este se reiniciará cuando algún otro proceso ejecute una operación signal.

EXCLUSIÓN MUTUA

Considere n procesos, identificados como P(i), los cuales necesitan todos acceder al mismo recurso. Cada proceso tiene una sección crítica que accede al recurso. En cada proceso se ejecuta un Wait(s) justo antes de entrar en su sección crítica. Si el valor de s pasa a ser negativo, el proceso se bloquea. Si el valor es 1, entonces se decrementa a 0 y el proceso entra en su sección crítica inmediatamente; dado que s ya no es positivo, ningún otro proceso será capaz de entrar en su sección crítica.
El semáforo se inicializa a 1. Así, el primer proceso que ejecute un sWait será capaz de entrar en su sección crítica inmediatamente, poniendo el valor de s a 0. Cualquier otro proceso que intente entrar en su sección crítica la encontrará ocupada y se bloqueará, poniendo el valor de s a -1. Cualquier número de procesos puede intentar entrar, de forma que cada intento insatisfactorio conllevará otro decremento del valor de s. Cuando el proceso que inicialmente entró en su sección crítica salga de ella, s se incrementa y uno de los procesos bloqueados (si hay alguno) se extrae de la lista de procesos bloqueados asociada con el semáforo y se pone en estado Listo. Cuando sea planificado por el sistema operativo, podrá entrar en la sección crítica.

5.4 Monitor

MONITOR CON SEÑAL

Un monitor es una colección de procedimientos, variables y estructuras de datos que se agrupan en un tipo especial de módulo o paquete. Los procesos pueden invocar los procedimientos de un monitor en el momento en que deseen, pero no pueden acceder directamente a las estructuras de datos internas del monitor desde procedimientos declarados afuera del monitor.

Los monitores poseen una propiedad especial que los hace útiles para lograr la exclusión mutua: sólo un proceso puede estar activo en un monitor en un momento dado. Los monitores son una construcción de lenguaje de programación, así que el compilador sabe que son especiales y puede manejar las llamadas a procedimientos de monitor de una forma diferente a como maneja otras llamadas a procedimientos. Por lo regular, cuando un proceso invoca un procedimiento de monitor, las primeras instrucciones del procedimiento verifican si hay algún otro proceso activo en ese momento dentro del monitor. Si así es, el proceso invocador se suspende hasta que el otro proceso

abandona el monitor. Si ningún otro proceso está usando el monitor, el proceso invocador puede entrar.

Es responsabilidad del compilador implementar la exclusión mutua en las entradas a monitores, pero una forma común es usar un semáforo binario. Puesto que el compilador, no el programador, se está encargando de la exclusión mutua, es mucho menos probable que algo salga mal. En cualquier caso, la persona que escribe el monitor no tiene que saber cómo el compilador logra la exclusión mutua; le basta con saber que si convierte todas las regiones críticas en procedimientos de monitor, dos procesos nunca podrán ejecutar sus regiones críticas al mismo tiempo.

Un monitor es un módulo software consistente en uno o más procedimientos, una secuencia de inicialización y datos locales. Las principales características de un monitor son las siguientes:

- Las variables locales de datos son sólo accesibles por los procedimientos del monitor y no por ningún procedimiento externo.
- Un proceso entra en el monitor invocando uno de sus procedimientos.
- Sólo un proceso puede estar ejecutando dentro del monitor al tiempo; cualquier otro proceso que haya invocado al monitor se bloquea, en espera de que el monitor quede disponible.

Al cumplir la disciplina de sólo un proceso al mismo tiempo, el monitor es capaz de proporcionar exclusión mutua fácilmente. Las variables de datos en el monitor sólo pueden ser accedidas por un proceso a la vez. Así, una estructura de datos compartida puede ser protegida colocándola dentro de un monitor. Si los datos en el monitor representan cierto recurso, entonces el monitor proporciona la función de exclusión mutua en el acceso al recurso.

Para ser útil para la programación concurrente, el monitor debe incluir herramientas de sincronización. Por ejemplo, suponga un proceso que invoca a un monitor y mientras está en él, deba bloquearse hasta que se satisfaga cierta condición. Se precisa una funcionalidad mediante la cual el proceso no sólo se bloquee, sino que libere el monitor para que algún otro proceso pueda entrar en él. Más tarde, cuando la condición se haya satisfecho y el monitor esté disponible nuevamente, el proceso debe poder ser retomado y permitida su entrada en el monitor en el mismo punto en que se suspendió.

Un monitor soporta la sincronización mediante el uso de variables condición que están contenidas dentro del monitor y son accesibles sólo desde el monitor. Las variables condición son un tipo de datos especial en los monitores que se manipula mediante dos funciones:

- **cwait(c)**: Suspende la ejecución del proceso llamante en la condición c. El monitor queda disponible para ser usado por otro proceso.
- **csignal(c**): Retoma la ejecución de algún proceso bloqueado por un cwait en la misma condición. Si hay varios procesos, elige uno de ellos; si no hay ninguno, no hace nada.

Nótese que las operaciones wait y signal de los monitores son diferentes de las de los semáforos. Si un proceso en un monitor señala y no hay ningún proceso esperando en la variable condición, la señal se pierde.

Aunque un proceso puede entrar en el monitor invocando cualquiera de sus procedimientos, puede entenderse que el monitor tiene un único punto de entrada que es el protegido, de ahí que sólo un proceso pueda estar en el monitor a la vez. Otros procesos que intenten entrar en el monitor se unirán a una cola de procesos bloqueados esperando por la disponibilidad del monitor. Una vez que un proceso está en el monitor, puede temporalmente bloquearse a sí mismo en la condición x realizando un cwait(x); en tal caso, el proceso será añadido a una cola de procesos esperando a reentrar en el monitor cuando cambie la condición y retomar la ejecución en el punto del programa que sigue a la llamada cwait(x).

Si un proceso que está ejecutando en el monitor detecta un cambio en la variable condición x, realiza un csignal(x), que avisa del cambio a la correspondiente cola de la condición.
La ventaja que los monitores tienen sobre los semáforos es que todas las funciones de sincronización están confinadas en el monitor. Por tanto, es más fácil comprobar que la sincronización se ha realizado correctamente y detectar los errores. Es más, una vez un monitor se ha programado correctamente, el acceso al recurso protegido será correcto para todo acceso desde cualquier proceso. En cambio, con los semáforos, el acceso al recurso será correcto sólo si todos los procesos que acceden al recurso han sido programados correctamente.

5.5 Paso de Mensajes

Los procesos necesitan ser sincronizados para conseguir exclusión mutua; los procesos cooperantes pueden necesitar intercambiar información. Un enfoque que proporciona ambas funciones es el paso de mensajes. El paso de mensajes tiene la ventaja añadida de que se presta a ser implementado tanto en sistemas distribuidos como en multiprocesadores de memoria compartida y sistemas monoprocesador.

La funcionalidad real del paso de mensajes se proporciona normalmente en forma de un par de primitivas:

- **send**(destino, mensaje)
- **receive**(origen, mensaje)

SINCRONIZACIÓN

La comunicación de un mensaje entre dos procesos implica cierto nivel de sincronización entre los dos: el receptor no puede recibir un mensaje hasta que no lo haya enviado otro proceso. En suma,

tenemos que especificar qué le sucede a un proceso después de haber realizado una primitiva send o receive.
Así, ambos emisor y receptor pueden ser bloqueantes o no bloqueantes. Tres son las combinaciones típicas, si bien un sistema en concreto puede normalmente implementar sólo una o dos de las combinaciones:

- Envío bloqueante, recepción bloqueante. Ambos emisor y receptor se bloquean hasta que el mensaje se entrega; a esto también se le conoce normalmente como rendezvous.
- Envío no bloqueante, recepción bloqueante. Aunque el emisor puede continuar, el receptor se bloqueará hasta que el mensaje solicitado llegue. Esta es probablemente la combinación más útil.
- Envío no bloqueante, recepción no bloqueante. Ninguna de las partes tiene que esperar.

DIRECCIONAMIENTO

Claramente, es necesario tener una manera de especificar en la primitiva de envío qué procesos deben recibir el mensaje. De igual modo, la mayor parte de las implementaciones permiten al proceso receptor indicar el origen del mensaje a recibir.
Los diferentes esquemas para especificar procesos en las primitivas send y receive caben dentro de dos categorías: **direccionamiento directo** y **direccionamiento indirecto.** Con el direccionamiento directo, la primitiva send incluye un identificador específico del proceso destinatario. La primitiva receive puede ser manipulada de dos maneras. Una posibilidad es que el proceso deba designar explícitamente un proceso emisor. Así, el proceso debe conocer con anticipación de qué proceso espera el mensaje. Esto suele ser lo más eficaz para procesos concurrentes cooperantes. En otros casos, sin embargo, es imposible especificar con anticipación el proceso de origen. Un ejemplo es un proceso servidor de impresión, que deberá aceptar un mensaje de solicitud de impresión de cualquier otro proceso. Para tales aplicaciones, una solución más efectiva es el uso de direccionamiento implícito. En este caso, el parámetro origen de la primitiva receive toma un valor devuelto por la operación de recepción cuando se completa.

El otro esquema general es el direccionamiento indirecto. En este caso, los mensajes no se envían directamente por un emisor a un receptor sino que son enviados a una estructura de datos compartida que consiste en colas que pueden contener mensajes temporalmente. Tales colas se conocen generalmente como buzones (mailboxes). Así, para que dos procesos se comuniquen, un proceso envía un mensaje al buzón apropiado y otro proceso toma el mensaje del buzón.

Una virtud del uso del direccionamiento indirecto es que, desacoplando emisor y receptor, se permite una mayor flexibilidad en el uso de mensajes. La relación entre emisores y receptores puede ser uno-a-uno, muchos-a-uno, uno-a-muchos o muchos-a-muchos. Una relación uno-a-uno permite establecer un enlace de comunicaciones privadas entre dos procesos. Esto aísla su interacción de interferencias erróneas de otros procesos. Una relación muchos-a-uno es útil para interacciones cliente/servidor; un proceso proporciona servicio a otros muchos procesos. En este caso, el buzón se conoce normalmente como puerto. Una relación uno-a-muchos permite un emisor y múltiples

receptores; esto es útil para aplicaciones donde un mensaje, o cierta información, debe ser difundido a un conjunto de procesos. Una relación muchos-a-muchos permite a múltiples procesos servidores proporcionar servicio concurrente a múltiples clientes.

La asociación de procesos a buzones puede ser estática o dinámica. Normalmente los puertos se asocian estáticamente con un proceso en particular; esto es, el puerto se crea y asigna a un proceso permanentemente. De igual modo, normalmente una relación uno-a-uno se define estática y permanentemente. Cuando hay varios emisores, la asociación de un emisor a un buzón puede ocurrir dinámicamente. Para este propósito pueden utilizarse primitivas como connect y disconnect.

Un aspecto relacionado tiene que ver con la propiedad del buzón. En el caso de un puerto, típicamente es creado por (y propiedad de) el proceso receptor. Así, cuando se destruye el proceso, el puerto también. Para el caso general de buzón, el sistema operativo puede ofrecer un servicio para crearlos. Tales buzones pueden verse bien como propiedad del proceso que lo crea, en cuyo caso se destruye junto con el proceso, o bien propiedad del sistema operativo, en cuyo caso se precisa un mandato explícito para destruir el buzón.

FORMATO DE MENSAJE

El formato del mensaje depende de los objetivos de la facilidad de mensajería y de cuándo tal facilidad ejecuta en un computador único o en un sistema distribuido. En algunos sistemas operativos, los diseñadores han preferido mensajes cortos de longitud fija para minimizar la sobrecarga de procesamiento y almacenamiento. Si se va a transferir una gran cantidad de datos, los datos pueden estar dispuestos en un archivo y el mensaje puede simplemente indicar el archivo. Una solución más sencilla es permitir mensajes de longitud variable. El mensaje está dividido en dos partes: una cabecera, que contiene información acerca del mensaje, y un cuerpo, que contiene el contenido real del mensaje.

DISCIPLINA DE COLA

La disciplina de cola más simple es FIFO, pero puede no ser suficiente si algunos mensajes son más urgentes que otros. Una alternativa es permitir especificar la prioridad del mensaje, en base al tipo de mensaje o por indicación del emisor. Otra alternativa es permitir al receptor inspeccionar la cola de mensajes y seleccionar qué mensaje quiere recibir el siguiente.

6 Concurrencia. Interbloqueo e inanición

6.1 Interbloqueos

Se puede definir el interbloqueo como el bloqueo permanente de un conjunto de procesos que o bien compiten por recursos del sistema o se comunican entre sí. Un conjunto de procesos está interbloqueado cuando cada proceso del conjunto está bloqueado esperando un evento (normalmente la liberación de algún recurso requerido) que sólo puede generar otro proceso bloqueado del conjunto. El interbloqueo es permanente porque no puede producirse ninguno de los eventos.

LAS CONDICIONES PARA EL INTERBLOQUEO

Deben presentarse tres condiciones de gestión para que sea posible un interbloqueo:

1. Exclusión mutua. Sólo un proceso puede utilizar un recurso en cada momento. Ningún proceso puede acceder a una unidad de un recurso que se ha asignado a otro proceso.
2. Retención y espera. Un proceso puede mantener los recursos asignados mientras espera la asignación de otros recursos.
3. Sin expropiación. No se puede forzar la expropiación de un recurso a un proceso que lo posee.

Si se cumplen estas tres condiciones se puede producir un interbloqueo, pero aunque se cumplan puede que no lo haya. Para que realmente se produzca el interbloqueo, se requiere una cuarta condición:

4. Espera circular. Existe una lista cerrada de procesos, de tal manera que cada proceso posee al menos un recurso necesitado por el siguiente proceso de la lista.

Las tres primeras condiciones son necesarias pero no suficientes para que exista un interbloqueo. La cuarta condición es, realmente, una consecuencia potencial de las tres primeras

6.2 Prevención del Interbloqueo

La estrategia de prevención del interbloqueo consiste, de forma simplificada, en diseñar un sistema de manera que se excluya la posibilidad del interbloqueo. Se pueden clasificar los métodos de prevención del interbloqueo en dos categorías. Un método indirecto de prevención del interbloqueo es impedir la aparición de una de las tres condiciones necesarias listadas previamente (las tres primeras). Un método directo de prevención del interbloqueo impide que se produzca una espera circular (cuarta condición).

EXCLUSIÓN MUTUA

En general, la primera de las cuatro condiciones no puede eliminarse.

RETENCIÓN Y ESPERA

La condición de retención y espera puede eliminarse estableciendo que un proceso debe solicitar al mismo tiempo todos sus recursos requeridos, bloqueándolo hasta que se le puedan conceder simultáneamente todas las peticiones. Esta estrategia es insuficiente en dos maneras. En primer lugar, un proceso puede quedarse esperando mucho tiempo hasta que todas sus solicitudes de recursos puedan satisfacerse, cuando, de hecho, podría haber continuado con solamente algunos de los recursos. En segundo lugar, los recursos asignados a un proceso pueden permanecer inutilizados durante un periodo de tiempo considerable, durante el cual se impide su uso a otros procesos. Otro problema es que un proceso puede no conocer por anticipado todos los recursos que requerirá.

SIN EXPROPIACIÓN

Esta condición se puede impedir de varias maneras. En primer lugar, si a un proceso que mantiene varios recursos se le deniega una petición posterior, ese proceso deberá liberar sus recursos originales y, si es necesario, los solicitará de nuevo junto con el recurso adicional. Alternativamente, si un proceso solicita un recurso que otro proceso mantiene actualmente, el sistema operativo puede expropiar al segundo proceso y obligarle a liberar sus recursos.

ESPERA CIRCULAR

La condición de espera circular se puede impedir definiendo un orden lineal entre los distintos tipos de recursos. Si a un proceso le han asignado recursos de tipo R, posteriormente puede pedir sólo aquellos recursos cuyo tipo tenga un orden posterior al de R.

6.3 Predicción de Interbloqueos

En la prevención del interbloqueo, se restringen las solicitudes de recurso para impedir al menos una de las cuatro condiciones de interbloqueo.

Con la predicción del interbloqueo, se decide dinámicamente si la petición actual de reserva de un recurso, si se concede, podrá potencialmente causar un interbloqueo. La predicción del interbloqueo, por tanto, requiere el conocimiento de las futuras solicitudes de recursos del proceso. En esta sección se describen dos técnicas para predecir el interbloqueo:

- No iniciar un proceso si sus demandas podrían llevar al interbloqueo.
- No conceder una petición adicional de un recurso por parte de un proceso si esta asignación podría provocar un interbloqueo.

6.4 Detección de Interbloqueos

Con la detección del interbloqueo, los recursos pedidos se conceden a los procesos siempre que sea posible. Periódicamente, el sistema operativo realiza un algoritmo que le permite detectar la condición de espera circular.

La comprobación de si hay interbloqueo se puede hacer con tanta frecuencia como una vez por cada petición de recurso o, con menos frecuencia, dependiendo de la probabilidad de que ocurra un interbloqueo.

RECUPERACIÓN

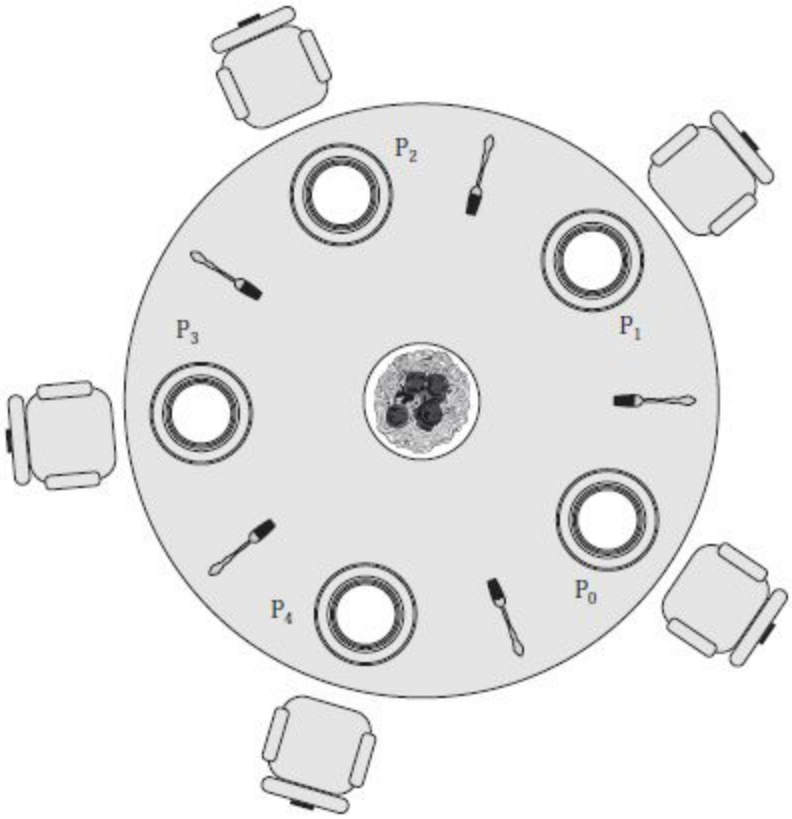
Una vez que se ha detectado el interbloqueo, se necesita alguna estrategia para recuperarlo. Las siguientes estrategias, listadas en orden de sofisticación creciente, son posibles:

1. Abortar todos los procesos involucrados en el interbloqueo.
2. Retroceder cada proceso en interbloqueo a algún punto de control (checkpoint) previamente definido, y rearrancar todos los procesos.
3. Abortar sucesivamente los procesos en el interbloqueo hasta que éste deje de existir.
4. Expropiar sucesivamente los recursos hasta que el interbloqueo deje de existir.

6.6 El problema de los filósofos y los comensales

Cinco filósofos viven en una casa, donde hay una mesa preparada para ellos. Básicamente, la vida de cada filósofo consiste en pensar y comer, y después de años de haber estado pensando, todos los filósofos están de acuerdo en que la única comida que contribuye a su fuerza mental son los espaguetis. Debido a su falta de habilidad manual, cada filósofo necesita dos tenedores para comer los espaguetis.

La disposición para la comida es simple: una mesa redonda en la que está colocado un gran cuenco para servir espaguetis, cinco platos, uno para cada filósofo, y cinco tenedores. Un filósofo que quiere comer se dirige a su sitio asignado en la mesa y, utilizando los dos tenedores situados a cada lado del plato, toma y come algunos espaguetis. El problema: diseñar un ritual (algoritmo) que permita a los filósofos comer. El algoritmo debe satisfacer la exclusión mutua (no puede haber dos filósofos que puedan utilizar el mismo tenedor a la vez) evitando el interbloqueo y la inanición (en este caso, el término tiene un sentido literal, además de algorítmico).



SOLUCIÓN UTILIZANDO SEMÁFOROS

Cada filósofo toma primero el tenedor de la izquierda y después el de la derecha. Una vez que el filósofo ha terminado de comer, vuelve a colocar los dos tenedores en la mesa. Esta solución, desgraciadamente, conduce al interbloqueo: Si todos los filósofos están hambrientos al mismo tiempo, todos ellos se sentarán, asirán el tenedor de la izquierda y tenderán la mano para tomar el otro tenedor, que no estará allí. En esta indecorosa posición, los filósofos pasarán hambre.

Como alternativa, se podría incorporar un asistente que sólo permitiera que haya cuatro filósofos al mismo tiempo en el comedor. Con un máximo de cuatro filósofos sentados, al menos un filósofo tendrá acceso a los dos tenedores.

7 Gestión de Memoria

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

Hardware básico. 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.

A la memoria se accede a través de una transacción del bus de memoria. El acceso a memoria puede requerir muchos ciclos de reloj del procesador para poderse completar, en cuyo caso el

procesador necesitara normalmente detenerse, ya que no dispondrá de los datos requeridos para completar la instrucción que este ejecutando. Esto es inadmisible debido a la gran frecuencia con la que la CPU accede a la memoria. La solución consiste en agregar una memoria rápida entre la CPU y la memoria principal, es un búfer de memoria y se denomina **cache**.
No solo debemos preocuparnos por esto, además debemos garantizar una correcta operación que proteja al sistema operativo de los posibles accesos por parta de procesos de usuarios y que también proteja a procesos de usuarios de otros. Debemos asegurarnos que cada proceso disponga de un espacio en memoria separado. Para esto, debemos poder determinar el rango de direcciones legales a las que el proceso puede acceder y garantizar que el proceso solo acceda a esas direcciones. Podemos proporcionar esta protección utilizando dos registros, el **registro base** almacena la dirección de memoria física legal más pequeña; y el **registro límite** que 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 la memoria de otros usuarios hará que se produzca una interrupción hacia el sistema operativo, que tratara dicho intento como un error fatal. Este esquema evita que un programa de usuario modifique (accidental o deliberadamente) el código y las estructuras de datos del sistema operativo o de otros usuarios.
Los registros base y limite solo pueden ser cargados por el sistema operativo, que utiliza una instrucción privilegiada especial. Esta instrucción solo puede ser ejecutada en modo kernel, y como solo el SO se ejecuta en modo kernel, únicamente este podrá cargar los registros base y limite. El SO, que se ejecuta en modo kernel, tiene acceso no restringido a la memoria tanto del propio SO como de los usuarios. Esto permite al sistema cargar los programas de usuario en la memoria de usuarios, validar dichos programas en caso de error, leer y modificar parámetros de las llamadas al sistema, etc.

Espacios 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 (es decir, la que se carga en el registro de direcciones de memoria de la memoria) se denomina comúnmente dirección física.
Los métodos de reasignación en el tiempo de compilación y en el tiempo de carga tienen 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 solamente 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 direcciónes lógicas es un espacio de direcciones físicas
La correspondencia entre direcciones y virtuales y físicas en tiempo de ejecución es establecida por un dispositivo de Hardware que se denomina unidad de gestión de memoria (MMU)

El programa de usuario nunca ve las direcciones físicas reales. El programa de usuario maneja direcciones lógicas y el hardware de conversión de memoria convierte esas direcciones lógicas en direcciones físicas.

7.1 Requisitos de la Gestión de Memoria

REUBICACIÓN

En un sistema multiprogramado, la memoria principal disponible se comparte generalmente entre varios procesos. Normalmente, no es posible que el programador sepa anticipadamente qué programas residirán en memoria principal en tiempo de ejecución de su programa. Adicionalmente, sería bueno poder intercambiar procesos en la memoria principal para maximizar la utilización del procesador, proporcionando un gran número de procesos para la ejecución. Una vez que un programa se ha llevado al disco, sería bastante limitante tener que colocarlo en la misma región de memoria principal donde se hallaba anteriormente, cuando éste se trae de nuevo a la memoria. Por el contrario, podría ser necesario reubicar el proceso a un área de memoria diferente.
Por tanto, no se puede conocer de forma anticipada dónde se va a colocar un programa y se debe permitir que los programas se puedan mover en la memoria principal, debido al intercambio o swap.

El sistema operativo necesitará conocer la ubicación de la información de control del proceso y de la pila de ejecución, así como el punto de entrada que utilizará el proceso para iniciar la ejecución. Debido a que el sistema operativo se encarga de gestionar la memoria y es responsable de traer el proceso a la memoria principal, estas direcciones son fáciles de adquirir. Adicionalmente, sin embargo, el procesador debe tratar con referencias de memoria dentro del propio programa. Las instrucciones de salto contienen una dirección para referenciar la instrucción que se va a ejecutar a continuación. Las instrucciones de referencia de los datos contienen la dirección del byte o palabra de datos referenciados. De alguna forma, el hardware del procesador y el software del sistema operativo deben poder traducir las referencias de memoria encontradas en el código del programa en direcciones de memoria físicas, que reflejan la ubicación actual del programa en la memoria principal.

El programa de usuario tendrá que recorrer varios pasos (algunos opcionales) antes de ser ejecutado. A lo largo de estos pasos, las direcciones pueden representarse de diferentes formas. Estas son generalmente simbólicas, un compilador se encarga de **reasignar** estas direcciones simbólicas a direcciones reubicables. El editor de montaje o cargador, se encargara de reasignar las direcciones reubicables a direcciones absolutas.

Dirección Simbólica	Dirección reubicable	Dirección Absoluta
“saldo”	*“14 bytes a partir de …”*	74014

La reasignación de las instrucciones y los datos a direcciones de memoria puede realizarse en cualquiera de los siguientes pasos:

· **Tiempo de compilación.** Si sabemos en el momento de realizar la compilación donde va residir el proceso en memoria, podremos generar código absoluto. Si la ubicación inicial cambiase en algún instante posterior, entonces sería necesario recompilar ese código.

· **Tiempo de carga.** Si no conocemos en tiempo de compilación donde va a residir el proceso en memoria, el compilador deberá generar código reubicable. La reasignación será entonces en el tiempo de carga, y si se realiza una modificación solo es necesario volver a cargar el código de usuario para incorporar el valor modificado.

· **Tiempo de Ejecución.** Si el proceso puede desplazarse durante su ejecución desde un segmento de memoria a otro, entonces es necesario retardar la reasignación hasta el instante de ejecución, para esto se precisara un hardware especial.

PROTECCIÓN

Cada proceso debe protegerse contra interferencias no deseadas por parte de otros procesos, sean accidentales o intencionadas. Por tanto, los programas de otros procesos no deben ser capaces de referenciar sin permiso posiciones de memoria de un proceso, tanto en modo lectura como escritura. Por un lado, lograr los requisitos de la reubicación incrementa la dificultad de satisfacer los requisitos de protección. Más aún, la mayoría de los lenguajes de programación permite el cálculo dinámico de direcciones en tiempo de ejecución (por ejemplo, calculando un índice de posición en un vector o un puntero a una estructura de datos). Por tanto, todas las referencias de memoria generadas por un proceso deben comprobarse en tiempo de ejecución para poder asegurar que se refieren sólo al espacio de memoria asignado a dicho proceso. Afortunadamente, se verá que los mecanismos que dan soporte a la reasignación también dan soporte al requisito de protección.

Normalmente, un proceso de usuario no puede acceder a cualquier porción del sistema operativo, ni al código ni a los datos. De nuevo, un programa de un proceso no puede saltar a una instrucción de otro proceso. Sin un trato especial, un programa de un proceso no puede acceder al área de datos de otro proceso. El procesador debe ser capaz de abortar tales instrucciones en el punto de ejecución.

Obsérvese que los requisitos de protección de memoria deben ser satisfechos por el procesador (hardware) en lugar del sistema operativo (software). Esto es debido a que el sistema operativo no puede anticipar todas las referencias de memoria que un programa hará.

COMPARTICIÓN

Cualquier mecanismo de protección debe tener la flexibilidad de permitir a varios procesos acceder a la misma porción de memoria principal. Por ejemplo, si varios programas están ejecutando el mismo programa, es ventajoso permitir que cada proceso pueda acceder a la misma copia del programa en lugar de tener su propia copia separada. Procesos que estén cooperando en la misma tarea podrían necesitar compartir el acceso a la misma estructura de datos. Por tanto, el sistema de gestión de la

memoria debe permitir el acceso controlado a áreas de memoria compartidas sin comprometer la protección esencial. De nuevo, se verá que los mecanismos utilizados para dar soporte a la reubicación soportan también capacidades para la compartición.

ORGANIZACIÓN LÓGICA

Casi invariablemente, la memoria principal de un computador se organiza como un espacio de almacenamiento lineal o unidimensional, compuesto por una secuencia de bytes o palabras. A nivel físico, la memoria secundaria está organizada de forma similar. Mientras que esta organización es similar al hardware real de la máquina, no se corresponde a la forma en la cual los programas se construyen normalmente. La mayoría de los programas se organizan en módulos, algunos de los cuales no se pueden modificar (sólo lectura, sólo ejecución) y algunos de los cuales contienen datos que se pueden modificar.

ORGANIZACIÓN FÍSICA

La organización del flujo de información entre la memoria principal y secundaria supone una de las preocupaciones principales del sistema. La responsabilidad para este flujo podría asignarse a cada programador en particular, pero no es practicable o deseable por dos motivos:

- La memoria principal disponible para un programa más sus datos podría ser insuficiente. En este caso, el programador debería utilizar una técnica conocida como superposición (overlaying), en la cual los programas y los datos se organizan de tal forma que se puede asignar la misma región de memoria a varios módulos, con un programa principal responsable para intercambiar los módulos entre disco y memoria según las necesidades. Incluso con la ayuda de herramientas de compilación, la programación con overlays malgasta tiempo del programador.
- En un entorno multiprogramado, el programador no conoce en tiempo de codificación cuánto espacio estará disponible o dónde se localizará dicho espacio.

Por tanto, está claro que la tarea de mover la información entre los dos niveles de la memoria debería ser una responsabilidad del sistema. Esta tarea es la esencia de la gestión de la memoria.

7.2 Particionamiento de la Memoria

PARTICIONAMIENTO FIJO

El esquema más simple para gestionar la memoria disponible es repartir la en regiones con límites fijos.

Una posibilidad consiste en hacer uso de particiones del mismo tamaño. En este caso, cualquier proceso cuyo tamaño es menor o igual que el tamaño de partición puede cargarse en cualquier partición disponible. Si todas las particiones están llenas y no hay ningún proceso en estado Listo o Ejecutando, el sistema operativo puede mandar a swap a un proceso de cualquiera de las particiones y cargar otro proceso, de forma que el procesador tenga trabajo que realizar.

Existen dos dificultades con el uso de las particiones fijas del mismo tamaño:

- Un programa podría ser demasiado grande para caber en una partición. En este caso, el programador debe diseñar el programa con el uso de overlays, de forma que sólo se necesite una porción del programa en memoria principal en un momento determinado. Cuando se necesita un módulo que no está presente, el programa de usuario debe cargar dicho módulo en la partición del programa, superponiéndolo (overlaying) a cualquier programa o datos que haya allí.
- La utilización de la memoria principal es extremadamente ineficiente. Cualquier programa, sin importar lo pequeño que sea, ocupa una partición entera. Este fenómeno, en el cual hay espacio interno malgastado debido al hecho de que el bloque de datos cargado es menor que la partición, se conoce con el nombre de **fragmentación interna.**

Ambos problemas se pueden mejorar, aunque no resolver, utilizando particiones de tamaño diferente Por ejemplo, los programas de 16 Mbytes se pueden acomodar sin overlays. Las particiones más pequeñas de 8 Mbytes permiten que los programas más pequeños se puedan acomodar sin menor fragmentación interna.

PARTICIONAMIENTO DINÁMICO

Con particionamiento dinámico, las particiones son de longitud y número variable. Cuando se lleva un proceso a la memoria principal, se le asigna exactamente tanta memoria como requiera y no más. El método comienza correctamente, pero finalmente lleva a una situación en la cual existen muchos huecos pequeños en la memoria. A medida que pasa el tiempo, la memoria se fragmenta cada vez más y la utilización de la memoria se decrementa. Este fenómeno se conoce como f**ragmentación externa**, indicando que la memoria que es externa a todas las particiones se fragmenta de forma incremental, por contraposición a lo que ocurre con la fragmentación interna, descrita anteriormente.

Una técnica para eliminar la fragmentación externa es la compactación: de vez en cuando, el sistema operativo desplaza los procesos en memoria, de forma que se encuentren contiguos y de este modo toda la memoria libre se encontrará unida en un bloque.

Debido a que la compactación de memoria consume una gran cantidad de tiempo, el diseñador del sistema operativo debe ser inteligente a la hora de decidir cómo asignar la memoria a los procesos (cómo eliminar los huecos). A la hora de cargar o intercambiar un proceso a la memoria principal, y siempre que haya más de un bloque de memoria libre de suficiente tamaño, el sistema operativo debe decidir qué bloque libre asignar. Tres algoritmos de colocación que pueden considerarse son: **Mejor-ajuste** escoge el bloque más cercano en tamaño a la petición. **Primer-ajuste** comienza a analizar la memoria desde el principio y escoge el primer bloque disponible que sea suficientemente grande. **Siguiente-ajuste** comienza a analizar la memoria desde la última colocación y elige el siguiente bloque disponible que sea suficientemente grande.

7.3 Paginación

La memoria principal se divide en porciones de tamaño fijo relativamente pequeños, y que cada proceso también se divide en porciones pequeñas del mismo tamaño fijo. A dichas porciones de proceso, conocidas como **páginas**, se les asigna porciones disponibles de memoria, conocidas como **marcos**, o marcos de páginas. Esta sección muestra que el espacio de memoria malgastado por cada proceso debido a la fragmentación interna corresponde sólo a una fracción de la última página de un proceso. No existe fragmentación externa.

En un momento dado, algunos de los marcos de la memoria están en uso y algunos están libres. El sistema operativo mantiene una lista de marcos libres.

Un registro base sencillo de direcciones no basta en esta ocasión. En su lugar, el sistema operativo mantiene una **tabla de páginas** por cada proceso. La tabla de páginas muestra la ubicación del marco por cada página del proceso. Dentro del programa, cada dirección lógica está formada por un número de página y un desplazamiento dentro de la página. Es importante recordar que en el caso de una partición simple, una dirección lógica es la ubicación de una palabra relativa al comienzo del programa; el procesador la traduce en una dirección física. Con paginación, la traducción de direcciones lógicas a físicas las realiza también el hardware del procesador. Ahora el procesador debe conocer cómo acceder a la tabla de páginas del proceso actual. Presentado como una dirección lógica (número de página, desplazamiento), el procesador utiliza la tabla de páginas para producir una dirección física (número de marco, desplazamiento).

Una tabla de páginas contiene una entrada por cada página del proceso, de forma que la tabla se indexe fácilmente por el número de página (iniciando en la página 0). Cada entrada de la tabla de páginas contiene el número del marco en la memoria principal, si existe, que contiene la página correspondiente. Adicionalmente, el sistema operativo mantiene una única lista de marcos libres de todos los marcos de la memoria que se encuentran actualmente no ocupados y disponibles para las páginas.

Por tanto vemos que la paginación simple, tal y como se describe aquí, es similar al particionamiento fijo. Las diferencias son que, con la paginación, las particiones son bastante pequeñas; un programa podría ocupar más de una partición; y dichas particiones no necesitan ser contiguas.

Para hacer este esquema de paginación conveniente, el **tamaño de página** y por tanto el tamaño del marco debe ser una potencia de 2. Con el uso de un tamaño de página potencia de 2, es fácil demostrar que la dirección relativa, que se define con referencia al origen del programa, y la dirección lógica, expresada como un número de página y un desplazamiento, es lo mismo.
Las consecuencias de utilizar un tamaño de página que es una potencia de 2 son dobles. Primero, el esquema de direccionamiento lógico es transparente al programador, al ensamblador y al montador.

Cada dirección lógica (número de página, desplazamiento) de un programa es idéntica a su dirección relativa. Segundo, es relativamente sencillo implementar una función que ejecute el hardware para llevar a cabo dinámicamente la traducción de direcciones en tiempo de ejecución.

Resumiendo, con la paginación simple, la memoria principal se divide en muchos marcos pequeños de igual tamaño. Cada proceso se divide en páginas de igual tamaño; los procesos más pequeños requieren menos páginas, los procesos mayores requieren más. Cuando un proceso se trae a la memoria, todas sus páginas se cargan en los marcos disponibles, y se establece una tabla de páginas. Esta técnica resuelve muchos de los problemas inherentes en el particionamiento.

7.4 Segmentacion

Un programa de usuario se puede subdividir utilizando segmentación, en la cual el programa y sus datos asociados se dividen en un número de **segmentos**. No se requiere que todos los programas sean de la misma longitud, aunque hay una longitud máxima de segmento. Como en el caso de la paginación, una dirección lógica utilizando segmentación está compuesta por dos partes, en este caso un número de segmento y un desplazamiento.

Debido al uso de segmentos de **distinto tamaño**, la segmentación es similar al particionamiento dinámico. En la ausencia de un esquema de overlays o el uso de la memoria virtual, se necesitaría que todos los segmentos de un programa se cargaran en la memoria para su ejecución. La diferencia, comparada con el particionamiento dinámico, es que con la segmentación un programa podría ocupar más de una partición, y estas particiones no necesitan ser contiguas. La segmentación elimina la fragmentación interna pero, al igual que el particionamiento dinámico, sufre de fragmentación externa. Sin embargo, debido a que el proceso se divide en varias piezas más pequeñas, la fragmentación externa debería ser menor.

Mientras que la paginación es invisible al programador, la segmentación es normalmente visible y se proporciona como una utilidad para organizar programas y datos. Normalmente, el programador o compilador asignará programas y datos a diferentes segmentos. Para los propósitos de la programación modular, los programas o datos se pueden dividir posteriormente en múltiples segmentos.

El inconveniente principal de este servicio es que el programador debe ser consciente de la limitación de tamaño de segmento máximo. Otra consecuencia de utilizar segmentos de distinto tamaño es que no hay una relación simple entre direcciones lógicas y direcciones físicas. De forma análoga a la paginación, un esquema de segmentación sencillo haría uso de una tabla de segmentos por cada proceso y una lista de bloques libre de memoria principal. Cada entrada de la tabla de segmentos tendría que proporcionar la dirección inicial de la memoria principal del correspondiente segmento. La entrada también debería proporcionar la longitud del segmento, para asegurar que no

se utilizan direcciones no válidas. Cuando un proceso entra en el estado Ejecutando, la dirección de su tabla de segmentos se carga en un registro especial utilizado por el hardware de gestión de la memoria.

Resumiendo, con la segmentación simple, un proceso se divide en un conjunto de segmentos que no tienen que ser del mismo tamaño. Cuando un proceso se trae a memoria, todos sus segmentos se cargan en regiones de memoria disponibles, y se crea la tabla de segmentos.

8 Memoria Virtual

La memoria virtual incluye la separación de la memoria lógica, tal como la conoce el usuario, con respecto a la memoria física. Esta separación permite proporcionar a los programadores una memoria virtual extremadamente grande, cuando solo está disponible una memoria fìsica mucho menor.

La memoria virtual facilita enormemente la tarea de programación, porque el programador ya no tiene que preocuparse de la cantidad de memoria física disponible. El espacio de direcciones virtuales de un proceso hace referencia a la forma lógica (o virtual) de almacenar un proceso en la memoria. Esta forma consiste en que el proceso comienza en una cierta dirección lógica y está almacenado en forma contigua en memoria, pero sin embargo, sabemos que la memoria física puede estar ordenada en marcos de página y que los marcos de página física asignados a un proceso pueden no ser contiguos. Es responsabilidad de la MMU establecer la correspondencia entre las páginas lógicas y los marcos de página física de la memoria.
Además, la memoria virtual también permite que dos o más procesos compartan los archivos y la memoria mediante mecanismos de compartición de páginas.

La idea básica detrás de la memoria virtual es que cada programa tiene su propio espacio de direcciones, el cual se divide en trozos llamados páginas. Cada página es un rango contiguo de direcciones. Estas páginas se asocian a la memoria física, pero no todas tienen que estar en la memoria física para poder ejecutar el programa. Cuando el programa hace referencia a una parte de su espacio de direcciones que está en la memoria física, el hardware realiza la asociación necesaria al instante. Cuando el programa hace referencia a una parte de su espacio de direcciones que no está en la memoria física, el sistema operativo recibe una alerta para buscar la parte faltante y volver a ejecutar la instrucción que falló.

8.1 Hardware y estructuras de control

Las dos características de la paginación y la segmentación que son la clave de este comienzo son:

1. Todas las referencias a la memoria dentro un proceso se realizan a direcciones lógicas, que se traducen dinámicamente en direcciones físicas durante la ejecución. Esto significa que un

proceso puede ser llevado y traído a memoria de forma que ocupe diferentes regiones de la memoria principal en distintos instantes de tiempo durante su ejecución.

2. Un proceso puede dividirse en varias porciones (páginas o segmentos) y estas porciones no tienen que estar localizadas en la memoria de forma contigua durante la ejecución. La combinación de la traducción de direcciones dinámicas en ejecución y el uso de una tabla de páginas o segmentos lo permite.

Ahora veamos cómo comenzar con la memoria dinámica. Si las dos características anteriores se dan, entonces es necesario que todas las páginas o todos los segmentos de un proceso se encuentren en la memoria principal durante la ejecución. Si la porción (segmento o página) en la que se encuentra la siguiente instrucción a buscar está y si la porción donde se encuentra la siguiente dirección de datos que se va a acceder también está, entonces al menos la siguiente instrucción se podrá ejecutar.

Usaremos el término porción para referirnos o bien a una página o un segmento, dependiendo si estamos empleando paginación o segmentación. Supongamos que se tiene que traer un nuevo proceso de memoria. El sistema operativo comienza trayendo únicamente una o dos porciones, que incluye la porción inicial del programa y la porción inicial de datos sobre la cual acceden las primeras instrucciones acceden. Esta parte del proceso que se encuentra realmente en la memoria principal para, cualquier instante de tiempo, se denomina conjunto residente del proceso. Cuando el proceso está ejecutándose, las cosas ocurren de forma suave mientras que todas las referencias a la memoria se encuentren dentro del conjunto residente. Usando una tabla de segmentos o páginas, el procesador siempre es capaz de determinar si esto es así o no. Si el procesador encuentra una dirección lógica que no se encuentra en la memoria principal, generará una interrupción indicando un fallo de acceso a la memoria. El sistema operativo coloca al proceso interrumpido en un estado de bloqueado y toma el control. Para que la ejecución de este proceso pueda reanudarse más adelante, el sistema operativo necesita traer a la memoria principal la porción del proceso que contiene la dirección lógica que ha causado el fallo de acceso. Con este fin, el sistema operativo realiza una petición de E/S, una lectura a disco. Después de realizar la petición de E/S, el sistema operativo puede activar otro proceso que se ejecute mientras el disco realiza la operación de E/S. Una vez que la porción solicitada se ha traído a la memoria principal, una nueva interrupción de E/S se lanza, dando control de nuevo al sistema operativo, que coloca al proceso afectado de nuevo en el estado Listo.

Existen dos implicaciones, la segunda más sorprendente que la primera, y ambas dirigidas a mejorar la utilización del sistema:
Pueden mantenerse un mayor número de procesos en memoria principal. Debido a que sólo vamos a cargar algunas de las porciones de los procesos a ejecutar, existe espacio para más procesos. Esto nos lleva a una utilización más eficiente del procesador porque es más probable que haya al menos uno o más de los numerosos procesos que se encuentre en el estado Listo, en un instante de tiempo concreto.

Un proceso puede ser mayor que toda la memoria principal. Se puede superar una de las restricciones fundamentales de la programación. Sin el esquema que hemos estado discutiendo, un programador debe estar realmente atento a cuánta memoria está disponible. Si el programa que está escribiendo es demasiado grande, el programador debe buscar el modo de estructurar el programa en fragmentos que pueden cargarse de forma separada con un tipo de estrategia de superposición (overlay). Con la memoria virtual basada en paginación o segmentación, este trabajo se delega al sistema operativo y al hardware. En lo que concierne al programador, él está trabajando con una memoria enorme, con un tamaño asociado al almacenamiento en disco. El sistema operativo automáticamente carga porciones de un proceso en la memoria principal cuando éstas se necesitan.

Debido a que un proceso ejecuta sólo en la memoria principal, esta memoria se denomina memoria real. Pero el programador o el usuario perciben una memoria potencialmente mucho más grande —la cual se encuentra localizada en disco. Esta última se denomina memoria virtual. La memoria virtual permite una multiprogramación muy efectiva que libera al usuario de las restricciones excesivamente fuertes de la memoria principal.

PAGINACIÓN

Para la memoria virtual basada en el esquema de paginación también se necesita una tabla de páginas. De nuevo, normalmente se asocia una única tabla de páginas a cada proceso. En este caso, sin embargo, las entradas de la tabla de páginas son más complejas. Debido a que sólo algunas de las páginas de proceso se encuentran en la memoria principal, se necesita que cada entrada de la tabla de páginas indique si la correspondiente página está presente (P) en memoria principal o no. Si el bit indica que la página está en memoria, la entrada también debe indicar el número de marco de dicha página.

La entrada de la tabla de páginas incluye un bit de modificado (M), que indica si los contenidos de la correspondiente página han sido alterados desde que la página se cargó por última vez en la memoria principal. Si no había ningún cambio, no es necesario escribir la página cuando llegue el momento de reemplazarla por otra página en el marco de página que actualmente ocupa. Pueden existir también otros bits de control en estas entradas. Por ejemplo, si la protección y compartición se gestiona a nivel de página, se necesitarán también los bits para este propósito.

Estructura de la tabla de páginas. El mecanismo básico de lectura de una palabra de la memoria implica la traducción de la dirección virtual, o lógica, consistente en un número de página y un desplazamiento, a la dirección física, consistente en un número de marco y un desplazamiento, usando para ello la tabla de páginas. Debido a que la tabla de páginas es de longitud variable dependiendo del tamaño del proceso, no podemos suponer que se encuentra almacenada en los registros. En lugar de eso, debe encontrarse en la memoria principal para poder ser accedida. Cuando un proceso en particular se encuentra ejecutando, un registro contiene la dirección de comienzo de la tabla de páginas para dicho proceso. El número de página de la dirección virtual se

utiliza para indexar esa tabla y buscar el correspondiente marco de página. Éste, combinado con la parte de desplazamiento de la dirección virtual genera la dirección real deseada.

En la mayoría de sistemas, existe una única tabla de página por proceso. Pero cada proceso puede ocupar una gran cantidad de memoria virtual. La cantidad de memoria demandada por las tablas de página únicamente puede ser inaceptablemente grande. Para resolver este problema, la mayoría de esquemas de memoria virtual almacena las tablas de páginas también en la memoria virtual, en lugar de en la memoria real. Esto representa que las tablas de páginas están sujetas a paginación igual que cualquier otra página. Cuando un proceso está en ejecución, al menos parte de su tabla de páginas debe encontrarse en memoria, incluyendo la entrada de tabla de páginas de la página actualmente en ejecución. Algunos procesadores utilizan un esquema de dos niveles para organizar las tablas de páginas de gran tamaño. En este esquema, existe un directorio de páginas, en el cual cada entrada apuntaba a una tabla de páginas.

Tabla de páginas invertida. Una desventaja del tipo de tablas de páginas que hemos visto es que su tamaño es proporcional al espacio de direcciones virtuales. Una estrategia alternativa al uso de tablas de páginas de uno o varios niveles es el uso de la estructura de tabla de páginas invertida.

En esta estrategia, la parte correspondiente al número de página de la dirección virtual se referencia por medio de un valor hash. El valor hash es un puntero para la tabla de páginas invertida, que contiene las entradas de tablas de página. Hay una entrada en la tabla de páginas invertida por cada marco de página real en lugar de uno por cada página virtual. De esta forma, lo único que se requiere para estas tablas de página siempre es una proporción fija de la memoria real, independientemente del número de procesos o de las páginas virtuales soportadas. Debido a que más de una dirección virtual puede traducirse en la misma entrada de la tabla hash, una técnica de encadenamiento se utiliza para gestionar el desbordamiento. Las técnicas de hashing proporcionan habitualmente cadenas que no son excesivamente largas —entre una y dos entradas. La estructura de la tabla de páginas se denomina invertida debido a que se indexan sus entradas de la tabla de páginas por el número de marco en lugar de por el número de página virtual.

La entrada en la tabla de páginas incluye la siguiente información:

- **Número de página**. Esta es la parte correspondiente al número de página de la dirección virtual.
- **Identificador del proceso**. El proceso que es propietario de está página. La combinación de número de página e identificador del proceso identifica a una página dentro del espacio de direcciones virtuales de un proceso en particular.
- **Bits de control**. Este campo incluye los flags, como por ejemplo, válido, referenciado, y modificado; e información de protección y cerrojos.

- **Puntero de la cadena**. Este campo es nulo (indicado posiblemente por un bit adicional) si no hay más entradas encadenadas en esta entrada. En otro caso, este campo contiene el valor del índice (número entre 0 y 2m-1) de la siguiente entrada de la cadena.

TLB (Buffer de traducción anticipada). En principio, toda referencia a la memoria virtual puede causar dos accesos a memoria física: uno para buscar la entrada la tabla de páginas apropiada y otro para buscar los datos solicitados. De esa forma, un esquema de memoria virtual básico causaría el efecto de duplicar el tiempo de acceso a la memoria. Para solventar este problema, la mayoría de esquemas de la memoria virtual utilizan una cache especial de alta velocidad para las entradas de la tabla de página, habitualmente denominada buffer de traducción anticipada (translation lookaside buffer - TLB). Esta cache funciona de forma similar a una memoria cache general y contiene aquellas entradas de la tabla de páginas que han sido usadas de forma más reciente. Dada una dirección virtual, el procesador primero examina la TLB, si la entrada de la tabla de páginas solicitada está presente (acierto en TLB), entonces se recupera el número de marco y se construye la dirección real. Si la entrada de la tabla de páginas solicitada no se encuentra (fallo en la TLB), el procesador utiliza el número de página para indexar la tabla de páginas del proceso y examinar la correspondiente entrada de la tabla de páginas. Si el bit de presente está puesto a 1, entonces la página se encuentra en memoria principal, y el procesador puede recuperar el número de marco desde la entrada de la tabla de páginas para construir la dirección real. El procesador también autorizará la TLB para incluir esta nueva entrada de tabla de páginas. Finalmente, si el bit presente no está puesto a 1, entonces la página solicitada no se encuentra en la memoria principal y se produce un fallo de acceso memoria, llamado fallo de página. En este punto, abandonamos el dominio del hardware para invocar al sistema operativo, el cual cargará la página necesaria y actualizada de la tabla de páginas.

Si una página solicitada no se encuentra en la memoria principal, una interrupción de fallo de página hace que se invoque a la rutina de tratamiento de dicho fallo de página. Para mantener la simplicidad de este diagrama, no se ha mostrado el hecho de que el sistema operativo pueda activar otro proceso mientras la operación de E/S sobre disco se está realizando. Debido al principio de proximidad, la mayoría de referencias a la memoria virtual se encontrarán situadas en una página recientemente utiliza y por tanto, la mayoría de referencias invocarán una entrada de la tabla de páginas que se encuentra en la cache.

Debido a que la TLB sólo contiene algunas de las entradas de toda la tabla de páginas, no es posible indexar simplemente la TLB por medio de número página. En lugar de eso, cada entrada de la TLB debe incluir un número de página así como la entrada de la tabla de páginas completa. El procesador proporciona un hardware que permite consultar simultáneamente varias entradas para determinar si hay una conciencia sobre un número de página. Esta técnica se denomina resolución asociativa (asociative mapping) que contrasta con la resolución directa, o indexación, utilizada para buscar en la tabla de páginas en la Figura 8.9. El diseño de la TLB debe considerar también la forma mediante la cual las entradas se organizan en ella y qué entrada se debe reemplazar cuando se necesite traer una nueva entrada.

Para concluir, el mecanismo de memoria virtual debe interactuar con el sistema de cache (no la cache de TLB, sino la cache de la memoria principal). Esto se ilustra en la Figura 8.10. Una dirección virtual tendrá generalmente el formato número de página, desplazamiento. Primero, el sistema de memoria consulta la TLB para ver si se encuentra presente una entrada de tabla de página que coincide. Si es así, la dirección real (física) se genera combinando el número de marco con el desplazamiento. Si no, la entrada se busca en la tabla de páginas. Una vez se ha generado la dirección real, que mantiene el formato de etiqueta y resto (remainder), se consulta la cache para ver si el bloque que contiene esa palabra se encuentra ahí. Si es así, se le devuelve a la CPU. Si no, la palabra se busca en la memoria principal.

La dirección virtual se traduce a una dirección real lo cual implica una referencia a la entrada de la tabla de páginas, que puede estar en la TLB, en la memoria principal, o en disco. La palabra referenciada puede estar en la cache, en la memoria principal, o en disco. Si dicha palabra referenciada se encuentra únicamente en disco, la página que contiene dicha palabra debe cargarse en la memoria principal y su bloque en la cache. Adicionalmente, la entrada en la tabla de páginas para dicha página debe actualizarse.

SEGMENTACIÓN

Las implicaciones en la memoria virtual. La segmentación permite al programador ver la memoria como si se tratase de diferentes espacios de direcciones o segmentos. Los segmentos pueden ser de tamaños diferentes, en realidad de tamaño dinámico. Una referencia a la memoria consiste en un formato de dirección del tipo (número de segmento, desplazamiento). Esta organización tiene un gran número de ventajas para el programador sobre los espacios de direcciones no segmentados:

1. Simplifica el tratamiento de estructuras de datos que pueden crecer. Si el programador no conoce a priori el tamaño que una estructura de datos en particular puede alcanzar es necesario hacer una estimación salvo que se utilicen tamaños de segmento dinámicos. Con la memoria virtual segmentada, a una estructura de datos se le puede asignar su propio segmento, y el sistema operativo expandirá o reducirá el segmento bajo demanda. Si un segmento que necesita expandirse se encuentre en la memoria principal y no hay suficiente tamaño, el sistema operativo poder mover el segmento a un área de la memoria principal mayor, si se encuentra disponible, o enviarlo a swap. En este último caso el segmento al que se ha incrementado el tamaño volverá a la memoria principal en la siguiente oportunidad que tenga.
2. Permite programas que se modifican o recopilan de forma independiente, sin requerir que el conjunto completo de programas se re-enlacen y se vuelvan a cargar. De nuevo, esta posibilidad se puede articular por medio de la utilización de múltiples segmentos.
3. Da soporte a la compartición entre procesos. El programador puede situar un programa de utilidad o una tabla de datos que resulte útil en un segmento al que pueda hacerse referencia desde otros procesos.

4. Soporta los mecanismos de protección. Esto es debido a que un segmento puede definirse para contener un conjunto de programas o datos bien descritos, el programador o el administrador de sistemas puede asignar privilegios de acceso de una forma apropiada.

Organización. En la exposición de la segmentación sencilla, indicamos que cada proceso tiene su propia tabla de segmentos, y que cuando todos estos segmentos se han cargado en la memoria principal, la tabla de segmentos del proceso se crea y se carga también en la memoria principal. Cada entrada de la tabla de segmentos contiene la dirección de comienzo del correspondiente segmento en la memoria principal, así como la longitud del mismo. El mismo mecanismo, una tabla segmentos, se necesita cuando se están tratando esquemas de memoria virtual basados en segmentación. De nuevo, lo habitual es que haya una única tabla de segmentos por cada uno de los procesos. En este caso sin embargo, las entradas en la tabla de segmentos son un poco más complejas (Figura 8.2b). Debido a que sólo algunos de los segmentos del proceso pueden encontrarse en la memoria principal, se necesita un bit en cada entrada de la tabla de segmentos para indicar si el correspondiente segmento se encuentra presente en la memoria principal o no. Si indica que el segmento está en memoria, la entrada también debe incluir la dirección de comienzo y la longitud del mismo.

Otro bit de control en la entrada de la tabla de segmentos es el bit de modificado, que indica si los contenidos del segmento correspondiente se han modificado desde que se cargó por última vez en la memoria principal. Si no hay ningún cambio, no es necesario escribir el segmento cuando se reemplace de la memoria principal. También pueden darse otros bits de control. Por ejemplo, si la gestión de protección y compartición se gestiona a nivel de segmento, se necesitarán los bits correspondientes a estos fines.

El mecanismo básico para la lectura de una palabra de memoria implica la traducción de una dirección virtual, o lógica, consistente en un número de segmento y un desplazamiento, en una dirección física, usando la tabla de segmentos. Debido a que la tabla de segmentos es de tamaño variable, dependiendo del tamaño del proceso, no se puede suponer que se encuentra almacenada en un registro. En su lugar, debe encontrarse en la memoria principal para poder accederse. Cuando un proceso en particular está en ejecución, un registro mantiene la dirección de comienzo de la tabla de segmentos para dicho proceso. El número de segmento de la dirección virtual se utiliza para indexar esta tabla y para buscar la dirección de la memoria principal donde comienza dicho segmento. Ésta es añadida a la parte de desplazamiento de la dirección virtual para producir la dirección real solicitada.

PAGINACIÓN Y SEGMENTACIÓN COMBINADAS

En un sistema combinado de paginación/segmentación, el espacio de direcciones del usuario se divide en un número de segmentos, a discreción del programador. Cada segmento es, por su parte, dividido en un número de páginas de tamaño fijo, que son del tamaño de los marcos de la memoria principal. Si un segmento tiene longitud inferior a una página, el segmento ocupará únicamente una

página. Desde el punto de vista del programador, una dirección lógica sigue conteniendo un número de segmento y un desplazamiento dentro de dicho segmento. Desde el punto de vista del sistema, el desplazamiento dentro del segmento es visto como un número de página y un desplazamiento dentro de la página incluida en el segmento.

Asociada a cada proceso existe una tabla de segmentos y varias tablas de páginas, una por cada uno de los segmentos. Cuando un proceso está en ejecución, un registro mantiene la dirección de comienzo de la tabla de segmentos de dicho proceso. A partir de la dirección virtual, el procesador utiliza la parte correspondiente al número de segmento para indexar dentro de la tabla de segmentos del proceso para encontrar la tabla de páginas de dicho segmento. Después, la parte correspondiente al número de página de la dirección virtual original se utiliza para indexar la tabla de páginas y buscar el correspondiente número de marco. Éste se combina con el desplazamiento correspondiente de la dirección virtual para generar la dirección real requerida.

Como antes, la entrada en la tabla de segmentos contiene la longitud del segmento. También contiene el campo base, que ahora hace referencia a la tabla de páginas. Los bits de presente y modificado no se necesitan debido a que estos aspectos se gestionan a nivel de página. Otros bits de control sí pueden utilizarse, a efectos de compartición y protección. La entrada en la tabla de páginas es esencialmente la misma que para el sistema de paginación puro. El número de página se proyecta en su número de marco correspondiente si la página se encuentra presente en la memoria. El bit de modificado indica si la página necesita escribirse cuando se expulse del marco de página actual. Puede haber otros bits de control relacionados con la protección u otros aspectos de la gestión de la memoria.

PROTECCIÓN Y COMPARTICIÓN

La segmentación proporciona una vía para la implementación de las políticas de protección y compartición. Debido a que cada entrada en la tabla de segmentos incluye la longitud así como la dirección base, un programa no puede, de forma descontrolada, acceder a una posición de memoria principal más allá de los límites del segmento. Para conseguir compartición, es posible que un segmento se encuentre referenciado desde las tablas de segmentos de más de un proceso. Los mecanismos están, por supuesto, disponibles en los sistemas de paginación. Sin embargo, en este caso la estructura de páginas de un programa y los datos no son visible para el programador, haciendo que la especificación de la protección y los requisitos de compartición sean menos cómodos.

8.2 Software del Sistema Operativo

POLÍTICA DE RECUPERACIÓN

La política de recuperación determina cuándo una página se trae a la memoria principal. Las dos alternativas habituales son bajo demanda y paginación adelantada (prepaging). Con paginación bajo

demanda, una página se trae a memoria sólo cuando se hace referencia a una posición en dicha página. Si el resto de elementos en la política de gestión de la memoria funcionan correctamente, ocurriría lo siguiente. Cuando un proceso se arranca inicialmente, va a haber una ráfaga de fallos de página. Según se van trayendo más y más páginas a la memoria, el principio de proximidad sugiere que las futuras referencias se encontrarán en las páginas recientemente traídas. Así, después de un tiempo, la situación se estabilizará y el número de fallos de página caerá hasta un nivel muy bajo.

Con paginación adelantada (prepaging), se traen a memoria también otras páginas, diferentes de la que ha causado el fallo de página. La paginación adelantada tiene en cuenta las características que tienen la mayoría de dispositivos de memoria secundaria, tales como los discos, que tienen tiempos de búsqueda y latencia de rotación. Si las páginas de un proceso se encuentran almacenadas en la memoria secundaria de forma contigua, es mucho más eficiente traer a la memoria un número de páginas contiguas de una vez, en lugar de traerlas una a una a lo largo de un periodo de tiempo más amplio. Por supuesto, esta política es ineficiente si la mayoría de las páginas que se han traído no se referencian a posteriori.

La política de paginación adelantada puede emplearse bien cuando el proceso se arranca, en cuyo caso el programador tiene que designar de alguna forma las páginas necesarias, o cada vez que ocurra un fallo de página. Este último caso es el más apropiado porque resulta completamente invisible al programador.

La paginación adelantada no se debe confundir con el swapping. Cuando un proceso se saca de la memoria y se le coloca en estado suspendido, todas sus páginas residentes se expulsan de la memoria. Cuando el proceso se recupera, todas las páginas que estaban previamente en la memoria principal retornan a ella.

POLÍTICA DE UBICACIÓN

La política de ubicación determina en qué parte de la memoria real van a residir las porciones de la memoria de un proceso. En los sistema de segmentación puros, la política de ubicación es un aspecto de diseño muy importante; políticas del estilo mejor ajuste, primer ajuste, y similares que se discutieron en el Capítulo 7, son las diferentes alternativas. Sin embargo, para sistemas que usan o bien paginación pura o paginación combinada con segmentación, la ubicación es habitualmente irrelevante debido a que el hardware de traducción de direcciones y el hardware de acceso a la memoria principal pueden realizar sus funciones en cualquier combinación de página-marco con la misma eficiencia.

POLÍTICA DE REEMPLAZO

Cuando todos los marcos de la memoria principal están ocupados y es necesario traer una nueva página para resolver un fallo de página, la política de reemplazo determina qué página de las que actualmente están en memoria va a reemplazarse. Todas las políticas tienen como objetivo que la

página que va a eliminarse sea aquella que tiene menos posibilidades de volver a tener una referencia en un futuro próximo.

Así, la mayoría de políticas tratan de predecir el comportamiento futuro en base al comportamiento pasado. En contraprestación, se debe considerar que cuanto más elaborada y sofisticada es una política de reemplazo, mayor va a ser la sobrecarga a nivel software y hardware para implementarla.

Es necesario mencionar una restricción que se aplica a las políticas de reemplazo antes de indagar en los diferentes algoritmos: algunos marcos de la memoria principal pueden encontrarse bloqueados. Cuando un marco está bloqueado, la página actualmente almacenada en dicho marco no puede reemplazarse. Gran parte del núcleo del sistema operativo se almacena en marcos que están bloqueados, así como otras estructuras de control claves.

Algoritmos Basicos.

- Sustitución FIFO. First in, first out, este algoritmo asocia con cada página el instante en que dicha página fue cargada en memoria. Cuando hace falta sustituir una página, se elige las más antigua. Este algoritmo también se puede manejar a través de una pila.
  Es un algoritmo fácil de entender y programar, pero no siempre eficiente. La página sustituida podría contener una variable muy utilizada que fue inicializada muy pronto y que se utiliza constantemente.
  Algo a tener en cuenta en este algoritmo es la **anomalía de Belady**, la cual ocurre cuando la tasa de fallos de página se incrementa a medida que se incrementa el número de marcos asignado, lo opuesto a lo que pensaríamos.
- Sustitución Óptima (OPT). Un algoritmo optimo será aquel que tenga la tasa más baja de fallos de página de entre todos los algoritmos y que nunca este sujeto a la anomalía de Belady. Consiste en remplazar la página que no vaya a ser usada durante el periodo de tiempo más largo. Este algoritmo es ideal y difícil de implementar ya que requiere un conocimiento futuro de la cadena de referencia. Se utiliza con propósitos comparativos.
- Sustitución LRU. Last recently used, menos recientemente usada. Este algoritmo asocia con cada página el instante correspondiente al último uso de dicha página. Cuando hay que sustituir una página, el algoritmo LRU, selecciona aquella que no haya sido utilizada durante el periodo más largo. Este algoritmo puede requerir una considerable asistencia de hardware, hay dos opciones: la utilización de contadores, por un lado; o manejándolo como una pila, por el otro. El algoritmo LRU no sufre la anomalía de Belady; al igual que el OPT, ambos pertenecen a un gripo de algoritmos llamados **algoritmos de pila**.
- Sustitución por aproximación LRU. Pocos sistemas informáticos proporcionan suficiente soporte de hardware como par implementar un verdadero algoritmo LRU, por lo tanto hay algoritmos aproximados, por ejemplo a través de un **bit de referencia**. Este es activado por hardware cada vez que se hace referencia a una página en particular. Los bits de referencia

están asociados a cada entrada de la tabla de páginas. Inicialmente todos están desactivados, si una página es utilizada se activa. De esta forma podemos saber que páginas han sido utilizadas y cuáles no. Esta es la base de muchos algoritmos de aproximación:

- Algoritmo de los Bits de referencia adicionales. Podemos disponer de información de ordenación adicional registrando los bits de referencia a intervalos regulares. Podemos mantener un byte de 8 bits para cada página en una tabla de memoria. Estos registros de 8 bits contienen el historial de uso de las páginas en los 8 últimos periodos temporales. Si interpretamos esos bytes de 8 bits como enteros sin signo, la página con el número más bajo será la menos recientemente usada y podremos sustituirla.
- Algoritmo de segunda oportunidad (reloj). Esencialmente es un algoritmo FIFO, pero cuando se selecciona una página, se mira el bit de referencia. Si el bit esta en 0, sustituimos dicha página, pero si esta en 1, damos a esa página una segunda oportunidad y seleccionamos la que sigue según FIFO. Cuando una página recibe una segunda oportunidad su bit de referencia se pone en 0 y se coloca el instante actual a su contador, de forma tal que no sea sustituida hasta que todas las demás lo hayan hecho, o en su defecto se les haya dado una segunda oportunidad también. Lo bueno es que si una página es utilizada con suficiente regularidad, su bit siempre estará activado y nunca será sustituida. Se utiliza usualmente una cola circular.
- Algoritmo de segunda oportunidad mejorado. Podemos mejorar el algoritmo considerando el bit de referencia y un bit de modificación como una pareja ordenada, lo que desencadena en los siguientes posibles casos:
  - (0,0) no se ha utilizado ni modificado recientemente. La página ideal para sustituir.
  - (0,1) no se ha usado pero si ha sido modificada. Sera necesario escribir la página antes de sustituirla.
  - (1,0) se ha utilizado reciénteme pero está limpia. Probablemente se la vuelva a usar pronto.
  - (1,1) se ha utilizado y se ha modificado recientemente. Probablemente se vuelva a utilizar y además sería necesario escribir la página antes de sustituirla. Es la peor opción para sustituir.

  Lo que se hace es ver a qué clase pertenece la página, y sustituimos la página que pertenece a la clase más baja.

- Sustitución basada en contador. Mantiene un contador del número de referencias que se hizo sobre la página, existen dos esquemas:
  - LFU. Least frequently used, menos frecuentemente usada. Requiere sustituir la página con el número de contador más bajo.

- MFU. Most frequently used, más frecuentemente usada. Se basa en que la pagina que tenga en contador más pequeño es porque probablemente recién se cargó, y aún necesita ser utiliza.

Buffering páginas. A pesar de que las políticas LRU y del reloj son superiores a FIFO, ambas incluyen una complejidad y una sobrecarga que FIFO no sufre. Adicionalmente, existe el aspecto relativo a que el coste de reemplazo de una página que se ha modificado es superior al de una que no lo ha sido, debido a que la primera debe escribirse en la memoria secundaria. Una estrategia interesante que puede mejorar el rendimiento de la paginación y que permite el uso de una política de reemplazo de páginas sencilla es el buffering de páginas. La estrategia más representativa de este tipo es la usaba por VAX VMS. El algoritmo de reemplazo de páginas es el FIFO sencillo. Para mejorar el rendimiento, una página remplazada no se pierde sino que se asigna a una de las dos siguientes listas: la lista de páginas libres si la página no se ha modificado, o la lista de páginas modificadas si lo ha sido. Véase que la página no se mueve físicamente de la memoria; al contrario, la entrada en la tabla de páginas para esta página se elimina y se coloca bien en la lista de páginas libres o bien en la lista de páginas modificadas. La lista de páginas libres es una lista de marcos de páginas disponibles para lectura de nuevas páginas.

ASIGNACIÓN DE MARCOS

Algoritmos de asignación. Una forma de asignar los marcos a los procesos, es repartir los *m* marcos entre *n* procesos de forma equitativa, es decir darle a cada proceso la misma cantidad de marcos (*m/n*). Esto se llama asignación equitativa.
Otro algoritmo es el de asignación proporcional, el cual asigna la memoria disponible de acuerdo al tamaño del proceso.

Asignación local y global.Podemos clasificar los algoritmos de planificación en dos categorías amplias: sustitución global y sustitución local. La sustitución global permite a un proceso seleccionar un marco de sustitución de entre el conjunto de todos los marcos, incluso si dicho marco está asignado actualmente a otro proceso; es decir, un proceso le puede quitar un marco a otro. El mecanismo de sustitución local requiere, por el contrario, que cada proceso solo efectué esa selección entre su propio conjunto de marcos asignados.
Un posible problema de la sustitución global es que un proceso no puede comprobar su propia tasa de fallos, ya que esta depende de cómo los procesos se disputaran los marcos. Pero la sustitución local ocasiona una baja en el rendimiento ya que un proceso queda acotado a los marcos que se le asignaron. El mecanismo de sustitución global da como resultado, generalmente, una mayor tasa de procesamiento del sistema y es por tanto el método que más comúnmente se usa.

SOBREPAGIANACIÒN

Si el número de marcos asignados a un proceso cae por debajo del número mínimo requerido por la arquitectura de la máquina, genera una alta tasa de paginación, lo que se denomina **sobrepaginación**. Un proceso entrara en sobrepaginación si invierte más tiempo implementando los mecanismos de paginación que en la propia ejecución del proceso.
Para limitar los efectos de la sobrepaginación podemos utilizar un algoritmo de sustitución local, de esta forma si uno de los procesos entra en sobrepaginación, no puede robar los marcos de otro proceso y hacer que este también entre en sobrepaginación. De todas formas no resuelve la sobrepaginación del primer proceso.
Una forma de prevenir la sobrepaginación es con la implementación del **modelo de localidad**. Este 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 compuesto, generalmente, por un conjunto de localidades diferentes. Las localidades están definidas por la estructura de programa y por sus estructuras de datos.

Frecuencia de fallos de página (PFF). Hay una estrategia más directa que está basada en la frecuencia de los fallos de página. Cuando la PFF es demasiada alta, sabemos que el proceso necesita más marcos; a la inversa, si la tasa es demasiada baja, puede que el proceso tenga signado demasiados marcos.
Podemos establecer límites superior e inferior, donde si la tasa real de fallos de página excede al límite superior, asignamos al proceso otro marco, mientras que si esa tasa cae por debajo del límite inferior, eliminamos un marco del proceso.

9 Planificación de Procesos

El objetivo de la planificación de procesos es asignar procesos a ser ejecutados por el procesador o procesadores a lo largo del tiempo, de forma que se cumplan los objetivos del sistema tales como el tiempo de respuesta, el rendimiento y la eficiencia del procesador. En muchos sistemas, esta actividad de planificación se divide en tres funciones independientes: planificación a largo-, medio-, y corto-plazo. Los nombres sugieren las escalas de tiempo relativas con que se ejecutan estas funciones.

La planificación a largo plazo se realiza cuando se crea un nuevo proceso. Hay que decidir si se añade un nuevo proceso al conjunto de los que están activos actualmente. La planificación a medio plazo es parte de la función de intercambio (swapping function). Hay que decidir si se añade un proceso a los que están al menos parcialmente en memoria y que, por tanto, están disponibles para su ejecución. La planificación a corto plazo conlleva decidir cuál de los procesos listos para ejecutar es ejecutado.

La planificación afecta al rendimiento del sistema porque determina qué proceso esperará y qué proceso progresará.

PLANIFICACIÓN A LARGO PLAZO

El planificador a largo plazo determina qué programas se admiten en el sistema para su procesamiento. De esta forma, se controla el grado de multiprogramación. Una vez admitido, un trabajo o programa de usuario se convierte en un proceso y se añade a la cola del planificador a corto plazo. En algunos sistemas, un proceso de reciente creación comienza en la zona de intercambio, en cuyo caso se añaden a la cola del planificador a medio plazo.

En un sistema por lotes, o en la parte de lotes de un sistema operativo de propósito general, los nuevos trabajos enviados se mandan al disco y se mantienen en una cola de lotes. El planificador a largo plazo creará procesos desde la cola siempre que pueda. En este caso hay que tomar dos decisiones. La primera, el planificador debe decidir cuándo el sistema operativo puede coger uno o más procesos adicionales. La segunda, el planificador debe decidir qué trabajo o trabajos se aceptan y son convertidos en procesos. Consideremos brevemente estas dos decisiones.

La decisión de cuándo crear un nuevo proceso se toma dependiendo del grado de multiprogramación deseado. Cuanto mayor sea el número de procesos creados, menor será el porcentaje de tiempo en que cada proceso se pueda ejecutar (es decir, más procesos compiten por la misma cantidad de tiempo de procesador). De esta forma, el planificador a largo plazo puede limitar el grado de multiprogramación a fin de proporcionar un servicio satisfactorio al actual conjunto de procesos. Cada vez que termine un trabajo, el planificador puede decidir añadir uno o más nuevos trabajos. Además, si la fracción de tiempo que el procesador está ocioso excede un determinado valor, se puede invocar al planificador a largo plazo. La decisión de qué trabajo admitir el siguiente, puede basarse en un sencillo «primero en llegar primero en servirse», o puede ser una herramienta para gestionar el rendimiento del sistema. El criterio utilizado puede incluir la prioridad, el tiempo estimado de ejecución y los requisitos de E/S. Por ejemplo, si la información está disponible, el planificador puede intentar encontrar un compromiso entre procesos limitados por el procesador y procesos limitados por la E/S2. Además, la decisión puede ser tomada dependiendo de los recursos de E/S que vayan a ser utilizados, de forma que se intente equilibrar el uso de la E/S.

Para los programas interactivos en un sistema de tiempo compartido, la petición de la creación de un proceso, puede estar generada por un usuario intentando conectarse al sistema. Los usuarios de tiempo compartido no se sitúan simplemente en una cola hasta que el sistema los pueda aceptar. Más exactamente, el sistema operativo aceptará a todos los usuarios autorizados hasta que el sistema se sature La saturación se estima utilizando ciertas medidas prefijadas del sistema. Llegado a este punto, una petición de conexión se encontrará con un mensaje indicando que el sistema está completo y el usuario debería reintentarlo más tarde.

PLANIFICACIÓN A MEDIO PLAZO

La planificación a medio plazo es parte de la función de intercambio. Con frecuencia, la decisión de intercambio se basa en la necesidad de gestionar el grado de multiprogramación. En un sistema que no utiliza la memoria virtual, la gestión de la memoria es también otro aspecto a tener en cuenta. De

esta forma, la decisión de meter un proceso en la memoria, tendrá en cuenta las necesidades de memoria de los procesos que están fuera de la misma.

PLANIFICACIÓN A CORTO PLAZO

En términos de frecuencia de ejecución, el planificador a largo plazo ejecuta con relativamente poca frecuencia y toma la decisión de grano grueso de admitir o no un nuevo proceso y qué proceso admitir. El planificador a medio plazo se ejecuta más frecuentemente para tomar decisiones de intercambio. El planificador a corto plazo, conocido también como activador, ejecuta mucho más frecuentemente y toma las decisiones de grano fino sobre qué proceso ejecutar el siguiente.

El planificador a corto plazo se invoca siempre que ocurre un evento que puede conllevar el bloqueo del proceso actual y que puede proporcionar la oportunidad de expulsar al proceso actualmente en ejecución en favor de otro.

PLANIFICACIÓN APROPIATIVA

Puede ser necesario tomar decisiones sobre planificación de la CPU en las siguientes circunstancias:

1. Cuando un proceso cambia del estado de ejecución al estado de espera.
2. Cuando un proceso cambia del estado de ejecución al estado de preparado.
3. Cuando un proceso cambia del estado de espera al estado preparado
4. Cuando un proceso termina.

En las situaciones 1 y 4 hay una sola opción: seleccionar un nuevo proceso (si hay en la cola). En las situaciones 2 y 3 existe la opción de planificar un nuevo proceso o no.

Cuando las decisiones de planificación solo tienen lugar en las circunstancias 1 y 4, decimos que el esquema de planificación es sin desalojo o cooperativo (en este caso, una vez que se ha asignado la CPU a un proceso, este se mantiene hasta que la CPU es liberada por la terminación del proceso o por la conmutación al estado de espera). En el caso contrario, decimos que se trata de un esquema apropiativo.

DESPACHADOR

Es el módulo que proporciona el control de la CPU a los procesos seleccionados por el planificador a corto plazo. Esta función implica:

- Cambio de contexto
- Cambio al modo usuario
- Salto a la posición correcta dentro del programa de usuario para reiniciar dicho programa

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

9.2 Algoritmos de planificación (Corto Plazo)

Primero en llegar, primero en servirse (first-come-first-served). La directiva de planificación más sencilla es primero en llegar primero en servirse (FCFS), también conocida como primero entra, primero-sale (FIFO) o como un sistema de colas estricto. En el momento en que un proceso pasa al estado de listo, se une a la cola de listos. Cuando el proceso actualmente en ejecución deja de ejecutar, se selecciona para ejecutar el proceso que ha estado más tiempo en la cola de listos. FCFS funciona mucho mejor para procesos largos que para procesos cortos.

El tiempo de estancia normalizado para el proceso Y es excesivamente grande en comparación con los otros procesos: el tiempo total que está en el sistema es 100 veces el tiempo requerido para su proceso. Esto sucederá siempre que llegue un proceso corto a continuación de un proceso largo. Por otra parte, incluso en este ejemplo extremo, los procesos largos no van mal. El proceso Z tiene un tiempo de estancia de casi el doble que Y, pero su tiempo de residencia normalizado está por debajo de 2,0.

Otro problema de FCFS es que tiende a favorecer procesos limitados por el procesador sobre los procesos limitados por la E/S. FCFS no es una alternativa atractiva por sí misma para un sistema uniprocesador. Sin embargo, a menudo se combina con esquemas de prioridades para proporcionar una planificación eficaz

Turno rotatorio (round robin). Una forma directa de reducir el castigo que tienen los trabajos cortos con FCFS es la utilización de expulsión basándose en el reloj. La política más sencilla es la del turno rotatorio, también denominada planificación cíclica. Se genera una interrupción de reloj cada cierto intervalo de tiempo. Cuando sucede la interrupción, el proceso actual en ejecución se sitúa en la cola de listos, y se selecciona el siguiente trabajo según la política FCFS. Esta técnica es también conocida como cortar el tiempo (time slicing), porque a cada proceso se le da una rodaja de tiempo antes de ser expulsado.

Con la planificación en turno rotatorio, el tema clave de diseño es la longitud del quantum de tiempo, o rodaja, a ser utilizada. Si el quantum es muy pequeño, el proceso se moverá por el sistema relativamente rápido. Por otra parte, existe una sobrecarga de procesamiento debido al manejo de la interrupción de reloj y por las funciones de planificación y activación. De esta forma, se deben evitar los quantums de tiempo muy pequeños. Una buena idea es que el quantum de tiempo debe ser ligeramente mayor que el tiempo requerido para una interacción o una función típica del proceso. Si es menor, muchos más procesos necesitarán, al menos, dos quantums de tiempo. La Figura 9.6 muestra el efecto que tiene en el tiempo de respuesta.

Nótese que en el caso extremo de un quantum de tiempo mayor que el proceso más largo en ejecución, la planificación en turno rotatorio degenera en FCFS.

La planificación en turno rotatorio es particularmente efectiva en sistemas de tiempo compartido de propósito general o en sistemas de procesamiento transaccional. Una desventaja de la planificación en turno rotatorio es que trata de forma desigual a los procesos limitados por el procesador y a los procesos limitados por la E/S. Generalmente, un proceso limitado por la E/S tiene ráfagas de procesador más cortas (cantidad de tiempo de ejecución utilizada entre operaciones de E/S) que los procesos limitados por el procesador. Si hay una mezcla de los dos tipos de procesos, sucederá lo siguiente: un proceso limitado por la E/S utiliza el procesador durante un periodo corto y luego se bloquea; espera a que complete la operación de E/S y a continuación se une a la cola de listos. Por otra parte, un proceso limitado por el procesador generalmente utiliza la rodaja de tiempo completa mientras ejecuta e inmediatamente vuelve a la cola de listos. De esta forma, los procesos limitados por el procesador tienden a recibir una rodaja no equitativa de tiempo de procesador, lo que conlleva un mal rendimiento de los procesos limitados por la E/S, uso ineficiente de los recursos de E/S y un incremento en la varianza del tiempo de respuesta.

Un refinamiento de la planificación en turno rotatorio es el que denomina turno rotatorio virtual (virtual round robin —VRR—) y que evita esta injusticia. Los nuevos procesos que llegan se unen a la cola de listos, que es gestionada con FCFS. Cuando expira el tiempo de ejecución de un proceso, vuelve a la cola de listos. Cuando se bloquea un proceso por E/S, se une a la cola de E/S. Hasta aquí, todo como siempre. La nueva característica es una cola auxiliar FCFS a la que se mueven los procesos después de estar bloqueados en una E/S. Cuando se va a tomar una decisión de activación, los procesos en la cola auxiliar tienen preferencia sobre los de la cola de listos. Cuando se activa un proceso desde la cola auxiliar, éste ejecuta por un tiempo no superior a la rodaja de tiempo, menos el tiempo total que ha estado ejecutando desde que no está en la cola principal de listos. Los estudios de rendimiento realizados por los autores, indican que este enfoque es realmente superior al turno rotatorio en términos de equidad.

Primero el proceso más corto (shortest process next SPN). Es una política no expulsiva en la que se selecciona el proceso con el tiempo de procesamiento más corto esperado. De esta forma un proceso corto se situará a la cabeza de la cola, por delante de los procesos más largos.

Sin embargo, se incrementa la variabilidad de los tiempos de respuesta, especialmente para los procesos más largos, y de esta forma se reduce la predecibilidad.

Un problema de la política SPN es la necesidad de saber, o al menos de estimar, el tiempo de procesamiento requerido por cada proceso. Para trabajos por lotes, el sistema puede requerir que el programador estime el valor y se lo proporcione al sistema operativo.

Un riesgo con SPN es la posibilidad de inanición para los procesos más largos, si hay una llegada constante de procesos más cortos. Por otra parte, aunque SPN reduce la predisposición a favor de los trabajos más largos, no es deseable para un entorno de tiempo compartido o de procesamiento transaccional por la carencia de expulsión.

Menor tiempo restante (shortest remaining time). La política del menor tiempo restante (SRT) es una versión expulsiva de SPN. En este caso, el planificador siempre escoge el proceso que tiene el menor tiempo de proceso restante esperado. Cuando un nuevo proceso se une a la cola de listos, podría tener un tiempo restante menor que el proceso actualmente en ejecución. Por tanto, el planificador podría expulsar al proceso actual cuando llega un nuevo proceso. Al igual que con SPN, el planificador debe tener una estimación del tiempo de proceso para realizar la función seleccionada, y existe riesgo de inanición para los procesos más largos.

SRT no tiene el sesgo en favor de los procesos largos que se puede encontrar en FCFS. A diferencia del turno rotatorio, no se generan interrupciones adicionales, reduciéndose la sobrecarga. Por otra parte, se deben almacenar los tiempos de servicio transcurridos, generando sobrecarga. SRT debería mejorar los tiempos de estancia de SPN, porque a un trabajo corto se le da preferencia sobre un trabajo más largo en ejecución.

Primero el de mayor tasa de respuesta (highest response ratio next). Para cada proceso individual, nos gustaría minimizar esta tasa, y nos gustaría minimizar el valor medio sobre todos los procesos. En general, no podemos saber por adelantado el tiempo de servicio de los procesos, pero lo podemos aproximar, basándonos en la historia anterior, en alguna entrada de usuario o en un gestor de configuración.

De esta forma, nuestra regla de planificación es como sigue: cuando se completa o bloquea el proceso actual, elegir el proceso listo con el mayor valor de R. Este enfoque es atractivo por que tiene en cuenta la edad del proceso. Mientras que se favorece a los procesos más cortos (un menor denominador lleva a una mayor tasa), el envejecimiento sin servicio incrementa la tasa, por lo que un proceso más largo podría competir con los trabajos más cortos.Como en SRT y SPN, el tiempo de servicio se debe estimar para usar la política primero el de mayor tasa de respuesta (HRRN).

Retroalimentación (feedback). Si no es posible averiguar el tiempo de servicio de varios procesos, SPN, SRT y HRRN no se pueden utilizar. Otra forma de establecer una preferencia para los trabajos más cortos es penalizar a los trabajos que han estado ejecutando más tiempo. En otras palabras, si no podemos basarnos en el tiempo de ejecución restante, nos podemos basar en el tiempo de ejecución utilizado hasta el momento.

La forma de hacerlo es la siguiente. La planificación se realiza con expulsión (por rodajas de tiempo), y se utiliza un mecanismo de prioridades dinámico. Cuando un proceso entra en el sistema, se sitúa en CL0 (véase Figura 9.4). Después de su primera expulsión, cuando vuelve al estado de Listo, se

sitúa en CL1. Cada vez que es expulsado, se sitúa en la siguiente cola de menor prioridad. Un proceso corto se completará de forma rápida, sin llegar a migrar muy lejos en la jerarquía de colas de listos. Un proceso más largo irá degradándose gradualmente. De esta forma, se favorece a los procesos nuevos más cortos sobre los más viejos y largos. Dentro de cada cola, excepto en la cola de menor prioridad, se utiliza un mecanismo FCFS. Una vez en la cola de menor prioridad, un proceso no puede descender más, por lo que es devuelto a esta cola repetidas veces hasta que se consigue completar. De esta forma, esta cola se trata con una política de turno rotatorio.

Este enfoque es conocido como retroalimentación multinivel, ya que el sistema operativo adjudica el procesador a un proceso y, cuando el proceso se bloquea o es expulsado, lo vuelve a situar en una de las varias colas de prioridad.

Planificación por prioridades. A cada proceso se le asocia una prioridad y la CPU se asigna al proceso que tenga la prioridad más alta. Los procesos con la misma prioridad se planifican como FCFS. Puede ser apropiativo o cooperativo. Cuando un proceso llega a la cola de procesos preparados, su prioridad se compara con la prioridad del proceso actualmente en ejecución. Un algoritmo de planificación por prioridades apropiativo expulsara de la CPU al proceso actual si la prioridad del proceso que acaba de llegar es mayor. Uno cooperativo simplemente pondrá el nuevo proceso al principio de la cola de procesos preparados.

Un problema importante de este tipo de algoritmo es el bloqueo indefinido o muerte por inanición, porque puede dejar a algunos procesos esperando indefinidamente y esto puede llevar a que finalmente el proceso se ejecuta en algún momento o bien el sistema termina fallando y se pierden los procesos no terminados. Una solución a este problema consiste en aplicar mecanismos de envejecimiento.

Planificación mediante colas multinivel. Divide la cola de procesos preparados en varias colas distintas. Los procesos se asignan permanentemente a una cola, generalmente en función de alguna propiedad del proceso, como por ejemplo el tamaño de memoria, la prioridad del proceso o tipo de proceso. Cada cola tiene su propio algoritmo de planificación y además debe definirse una planificación entre las colas, la cual suele implementarse como apropiativa y prioridad fija.

Planificación mediante colas multinivel realimentadas. Permite mover un proceso de una cola a otra. La idea es separar los procesos en función de las características de sus ráfagas de CPU. Si un proceso utiliza demasiado tiempo de CPU, se pasa a una cola de prioridad más baja (o al revés).

LA PLANIFICACIÓN CONTRIBUCIÓN JUSTA (FAIR-SHARE SCHEDULING)

Todos los algoritmos de planificación discutidos hasta el momento tratan a la colección de procesos listos como un simple conjunto de procesos de los cuales seleccionar el siguiente a ejecutar. Este conjunto de procesos se podría romper con el uso de prioridades, pero es homogéneo.

Sin embargo, en un sistema multiusuario, si las aplicaciones o trabajos de usuario se pueden organizar como múltiples procesos (o hilos), hay una estructura en la colección de procesos que no se reconoce en los planificadores tradicionales. Desde el punto de vista del usuario, la preocupación no es cómo ejecuta un simple proceso, sino cómo ejecutan su conjunto de procesos, que forman una aplicación. De esta forma, sería deseable tomar decisiones de planificación basándose en estos conjuntos de procesos. Este enfoque se conoce normalmente como planificación contribución justa. Además, el concepto se puede extender a un grupo de usuarios, incluso si cada usuario se representa con un solo proceso. Por ejemplo, en un sistema de tiempo compartido, podríamos querer considerar a todos los usuarios de un determinado departamento como miembros del mismo grupo. Se podrían tomar las decisiones de planificación intentando dar a cada grupo un servicio similar. De esta forma, si muchos usuarios de un departamento se meten en el sistema, nos gustaría ver cómo el tiempo de respuesta se degrada sólo para los miembros de este departamento, más que para los usuarios de otros departamentos.

El término contribución justa indica la filosofía que hay detrás de este planificador. A cada usuario se le asigna una prima de algún tipo que define la participación del usuario en los recursos del sistema como una fracción del uso total de dichos recursos. En particular, a cada usuario se le asigna una rodaja del procesador. Este esquema debería operar, más o menos, de forma lineal. Así, si un usuario A tiene el doble de prima que un usuario B, en una ejecución larga, el usuario A debería ser capaz de hacer el doble de trabajo que el usuario B. El objetivo del planificador contribución justa es controlar el uso para dar menores recursos a usuarios que se han excedido de su contribución justa y mayores recursos a los que no han llegado.

10. Archivos

Los archivos son un mecanismo de abstracción. Proporcionan una manera de almacenar información en el disco y leerla después. Esto se debe hacer de tal forma que se proteja al usuario de los detalles acerca de cómo y dónde se almacena la información y cómo funcionan los discos en realidad. Las reglas para denominar archivos varían de un sistema a otro, pero casi todos los sistemas operativos permiten cadenas de una a ocho letras como nombre. Por ende, casa, mesa y silla son posibles nombres de archivos. Con frecuencia, también se permiten dígitos y caracteres especiales, por lo que nombres como 2 y urgente! son a menudo válidos. Muchos sistemas de archivos admiten nombres de hasta 255 caracteres. Algunos sistemas de archivos diferencian las letras mayúsculas de las minúsculas, mientras que otros no. UNIX cae en la primera categoría; MS-DOS en la segunda. Así, un sistema UNIX puede tener los siguientes nombres como tres archivos distintos: maria, Maria

y MARIA. En MS-DOS, todos estos nombres se refieren al mismo archivo. Muchos sistemas operativos aceptan nombres de archivos en dos partes, separadas por un punto, como en prog.c. La parte que va después del punto se conoce como la **extensión del archivo** y por lo general indica algo acerca de su naturaleza. Por ejemplo, en MS-DOS, los nombres de archivos son de 1 a 8 caracteres, más una extensión opcional de 1 a 3 caracteres. En UNIX el tamaño de la extensión (si la hay) es a elección del usuario y un archivo puede incluso tener dos o más extensiones, como en paginainicio.html.zip, donde .html indica una página Web en HTML y .zip indica que el archivo se ha comprimido mediante el programa zip.

10.1 Atributos de archivos

Todo archivo tiene un nombre y sus datos. Además, todos los sistemas operativos asocian otra información con cada archivo; por ejemplo, la fecha y hora de la última modificación del archivo y su tamaño. A estos elementos adicionales les llamaremos **atributos del archivo** o **metadatos**. La lista de atributos varía considerablemente de un sistema a otro.

10.2 Operaciones de archivos

Los archivos existen para almacenar información y permitir que se recupere posteriormente. Distintos sistemas proveen diferentes operaciones para permitir el almacenamiento y la recuperación. Algunas de las operaciones más comunes se definen a continuación:

1) *CREATE:* El archivo se crea sin datos. El propósito de la llamada es anunciar la llegada del archivo y establecer algunos de sus atributos.
2) *DELETE:* Cuando el archivo ya no se necesita, se tiene que eliminar para liberar espacio en el disco.
3) *OPEN:* Antes de usar un archivo, un proceso debe abrirlo. El propósito de la llamada open es permitir que el sistema lleve los atributos y la lista de direcciones de disco a memoria principal para tener un acceso rápido a estos datos en llamadas posteriores.
4) *CLOSE:* Cuando terminan todos los accesos, los atributos y las direcciones de disco ya no son necesarias, por lo que el archivo se debe cerrar para liberar espacio en la tabla interna.
5) *READ:* Los datos se leen del archivo. Por lo general, los bytes provienen de la posición actual. El llamador debe especificar cuántos datos se necesitan y también debe proporcionar un búfer para colocarlos.
6) *WRITE:* Los datos se escriben en el archivo otra vez, por lo general en la posición actual. Si la posición actual es al final del archivo, aumenta su tamaño. Si la posición actual está en medio del archivo, los datos existentes se sobrescriben y se pierden para siempre.

10. 3 Tipos de archivos

Muchos SO soportan varios tipos de archivos. Por ejemplo, UNIX y Windows tienen archivos regulares. UNIX también tiene archivos especiales de caracteres y de bloques. Los archivos regulares son los que contienen información del usuario. Los archivos especiales de caracteres se relacionan con la entrada/salida y se utilizan para modelar dispositivos de E/S en serie, tales como terminales, impresoras y redes. Los archivos especiales de bloques se utilizan para modelar discos.

10.4 Métodos de acceso

Los primeros SO proporcionaban sólo un tipo de acceso: **acceso secuencial**. En estos sistemas, un proceso podía leer todos los bytes o registros en un archivo, empezando desde el principio, pero no podía saltar algunos y leerlos fuera de orden. Sin embargo, los archivos secuenciales podían rebobinarse para poder leerlos todas las veces que fuera necesario. Los archivos secuenciales eran convenientes cuando el medio de almacenamiento era cinta magnética en vez de disco.
Cuando se empezó a usar discos para almacenar archivos, se hizo posible leer los bytes o registros de un archivo fuera de orden. A estos archivos se los llama **archivos de acceso aleatorio.** Los archivos de acceso aleatorio son esenciales para muchas aplicaciones, como los sistemas de bases de datos. Por ejemplo, si el cliente de una aerolínea llama y desea reservar un asiento en un vuelo específico, el programa de reservación debe tener acceso al registro para ese vuelo sin tener que leer primero todos los miles de registros de los demás vuelos.

10.5 Administración del espacio en disco

Hay dos estrategias posibles para almacenar un archivo de n bytes: se asignan n bytes consecutivos de espacio en disco o el archivo se divide en varios bloques (no necesariamente) contiguos. Almacenar un archivo como una secuencia contigua de bytes tiene el problema de que, si un archivo crece, probablemente tendrá que moverse en el disco.

Asignación de espacio a archivos. En todos los sistemas, hay múltiples archivos almacenados en el mismo disco. El problema principal es como asignar el espacio a esos archivos de modo que el espacio de disco se utilice eficazmente y que se pueda acceder a esos archivos rápidamente. Para esto, existen diferentes métodos:

- La **asignación contigua** requiere que cada archivo ocupe un conjunto de bloques contiguos en el disco. De esta manera, suponiendo que se quiere acceder al disco, acceder al boque b+1 después del bloque b, no requiere ningún movimiento del cabezal. Por lo tanto, acceder a un archivo que haya sido asignado de manera contigua resulta sencillo.

· Con la **asignación enlazada,** cada archivo es una lista enlazada de bloques de disco, pudiendo estar dichos bloques por todo el disco. El directorio contiene un puntero al primer y al último bloque de cada archivo. Por ejemplo, un archivo de cinco bloques podría comenzar en el bloque 9 y continuar en el 16, luego el 1, después el 10 y finalizar en el 25. Cada bloque contiene un puntero al bloque siguiente. Dichos bloques no están a disposición del usuario. Para leer un archivo, simplemente leemos los bloques siguiendo los punteros de un bloque a otro.

Administración del espacio libre. Para controlar el espacio libre del disco, el sistema mantiene una lista de espacio libre. La **lista de espacio libre** indica todos los bloques de disco libres, es decir aquellos que no están asignados a ningún archivo o directorio. Para crear un archivo, exploramos la lista de espacio libre hasta obtener la cantidad de espacio requerida y asignamos ese espacio al nuevo archivo. A continuación, este espacio se elimina de la lista de espacio libre. Cuando se borra un archivo, su espacio de disco se añade a la lista de espacio libre.

Respaldo - Recuperación. Los archivos y directorios se mantienen tanto en memoria principal como en disco y se debe tener cuidado a que los fallos del sistema no provoquen una pérdida de datos. Con este fin, pueden utilizarse programas para realizar copias de seguridad de los datos de disco en otro dispositivo de almacenamiento. La recuperación de un archivo individual o de disco completo, puede ser entonces una cuestión de restaurar los datos a partir de la copia de seguridad.
Sistema con registro por diario (Journaling)
El Journaling es un mecanismo por el cual un sistema informático puede implementar transacciones. También se le conoce como registro por diario. Se basa en llevar un registro de diario en el que se almacena la información necesaria para restablecer los datos afectados por la transacción en caso de que esta falle.
Las aplicaciones más frecuentes de los sistemas de journaling se usan para implementar transacciones de sistemas de bases de datos y, más recientemente, para evitar la corrupción de las estructuras de datos en las que se basan los sistemas de archivos modernos.

11. Directorios

Los sistemas de archivos de las computadoras pueden tener una gran complejidad. Algunos sistemas almacenan millones de archivos en terabytes de disco. Para gestionar todos estos datos, necesitamos organizarlos de alguna manera y esta organización implica el uso de directorios.
El directorio puede considerarse como una tabla de símbolos que traduce los nombres de archivo a sus correspondientes entradas de directorio.
Operaciones que se realizan con el directorio:

- Búsqueda de un archivo.
- Crear un archivo.
- Borrar un archivo.
- Listar un directorio.

- Renombrar un archivo.
- Recorrer el sistema de archivos.

Directorio de un único nivel. Es la estructura de directorio más simple. Todos los archivos están contenidos en el mismo directorio, que resulta fácil de mantener y de comprender. Sin embargo, un directorio de un único nivel tiene limitaciones significativas cuando el número de archivos se incrementa o cuando el sistema tiene más de un usuario. Como todos los archivos están en el mismo directorio, deberán tener nombres distintivos.

Directorio de dos niveles. Cada usuario tiene su propio directorio de archivos de usuario (UFD). Cada uno de los UFD tiene una estructura similar, pero solo incluye los archivos de un único usuario.
Los propios directorios de usuario deben poder crearse y borrarse según sea necesario. Para ello se ejecuta un programa especial del sistema, con el nombre de usuario apropiado y la correspondiente información de la cuenta.
Este sistema aísla a un usuario y otro, lo que es una ventaja cuando los usuarios son totalmente independientes, pero una desventaja cuando los usuarios necesitan cooperar en cierta tarea y quieren acceder a los archivos de otro.
Directorio con estructura de árbol. Permite a los usuarios crear sus propios subdirectorios y organizar sus archivos correspondientemente. Los árboles son la estructura de directorio más común. Cada árbol tiene un directorio raíz y todos los archivos del sistema tienen un nombre de ruta distintivo.
Cada directorio (o subdirectorio) contiene un conjunto de archivos o subdirectorios. Un directorio es simplemente otro archivo, pero que se trata de forma especial.

Directorios en un grafo acíclico. Un grafo acíclico (es decir, un grafo sin ningún ciclo) permite que los directorios compartan subdirectorios de archivos. El mismo archivo o subdirectorio puede estar en dos directorios diferentes. Al ser compartido, solo existe un archivo real. Por lo que cualquier cambio que realizado por un usuario será inmediatamente visible para otro.

MONTAJE
Un sistema de archivos debe montarse para poder estar disponible para los procesos del sistema.
El proceso de montaje es bastante simple. Al sistema operativo se le proporciona el nombre de dispositivo y el punto de montaje que es la ubicación dentro de la estructura de archivos a la que hay que conectar el sistema de archivos que se está montando. Normalmente, el punto de montaje será un directorio vacío.
A continuación, el sistema operativo verifica que el dispositivo contiene un sistema de archivos valido y finalmente registra en su estructura de directorios que hay un sistema de archivos montado en el punto de montaje especificado.

Nombres de rutas. Cuando el sistema de archivos está organizado como un árbol de directorios, se necesita cierta forma de especificar los nombres de los archivos. Por lo general se utilizan dos

métodos distintos. En el primer método, cada archivo recibe un nombre de ruta absoluto que consiste en la ruta desde el directorio raíz al archivo. Los nombres de ruta absolutos siempre empiezan en el directorio raíz y son únicos.

Windows \usr\ast\mailbox
UNIX /usr/ast/mailbox
MULTICS >usr>ast>mailbox

El otro tipo de nombre es el nombre de ruta relativa. Éste se utiliza en conjunto con el concepto del directorio de trabajo (también llamado directorio actual). Un usuario puede designar un directorio como el directorio de trabajo actual, en cuyo caso todos los nombres de las rutas que no empiecen en el directorio raíz se toman en forma relativa al directorio de trabajo. Por ejemplo, si el directorio de trabajo actual es /usr/ast, entonces el archivo cuya ruta absoluta sea /usr/ast/mailbox se puede referenciar simplemente como mailbox. En otras palabras, el comando de UNIX

cp /usr/ast/mailbox /usr/ast/mailbox.bak

y el comando

cp mailbox mailbox.bak

Hacen exactamente lo mismo si el directorio de trabajo es /usr/ast. A menudo es más conveniente la forma relativa, pero hace lo mismo que la forma absoluta.

Cada proceso tiene su propio directorio de trabajo, por lo que cuando éste cambia y después termina ningún otro proceso se ve afectado y no quedan rastros del cambio en el sistema de archivos. De esta forma, siempre es perfectamente seguro para un proceso cambiar su directorio de trabajo cada vez que sea conveniente.

PROTECCIÓN

Cuando se almacena información en un sistema informático, necesitamos protegerla frente a los daños físicos (fiabilidad) y frente a los accesos incorrectos (protección).

La fiabilidad se proporciona, generalmente, mediante copias duplicadas de los archivos. Para proporcionar protección lo podemos hacer de varias formas.

Tipos de acceso. Los mecanismos de protección proporcionan un acceso controlado limitando los tipos de accesos a archivo que puedan realizarse. El acceso se permite o se deniega dependiendo de varios factores, uno de los cuales es el tipo de acceso solicitado. Podemos controlar varios tipos de operaciones diferentes: Lectura, Escritura, Ejecución, Adición, Borrado y Listado (de nombre y atributos).

Control de acceso. La técnica más común es hacer que el acceso dependa de la identidad del usuario. Los diferentes usuarios pueden necesitar diferentes tipos de acceso a un archivo o directorio. El esquema más general para hacer esto es asociar con cada archivo y directorio una lista de control de acceso que especifique los nombres de usuario y los tipos de acceso que se permiten para cada uno. Pero es una técnica tediosa y con muchas desventajas.

Para resolver este problema se usa una versión condensada de la lista de acceso clasificando a los usuarios en grupos:

Propietario: El que creo el archivo.

Grupo: Conjunto de usuarios que comparten el archivo y necesitan un acceso similar al mismo.
Universo: Todos los demás usuarios.

12. Estructura de almacenamiento masivo

12.1 Dispositivos

Discos magnéticos. Los discos magnéticos son hoy la principal estructura de almacenamiento masivo. Está conformado por uno o varios platos de forma circular plana. Las dos superficies de cada plato están recubiertas de un material magnético. La información se almacena grabándola magnéticamente sobre los platos.
Un cabezal de lectura-escritura "vuela" justo por encima de cada una de las superficies de cada plato. Los cabezales están conectados a un brazo de disco que mueve todos los cabezales como una misma unidad. La superficie de cada plato está dividida desde el punto de vista lógico en pistas circulares, que a su vez están divididas en sectores. El conjunto de las pistas que están situadas en una determinada posición del brazo forman un cilindro.
Cuando se está usando el disco, un motor lo hace girar a gran velocidad. La velocidad de un disco puede considerarse compuesta por dos partes diferenciadas:

- Velocidad de transferencia: es la velocidad con que los datos fluyen entre la computadora y la unidad de disco.
- Tiempo de posicionamiento o de acceso aleatorio: está compuesto por el tiempo necesario para mover el brazo de disco hasta el cilindro deseado, denominado tiempo de búsqueda, y el tiempo requerido para que el sector deseado rote hasta pasar por debajo del cabezal, denominado latencia rotacional

Como el cabezal "vuela" sobre un colchón de aire extremadamente fino, existe el peligro que el cabezal entre en contacto con la superficie del plato, este accidente se denomina aterrizaje de cabezales. Estos no pueden reparase, por lo que es necesario remplazar el disco completo.
Para conectar una unidad de disco a una computadora, se utiliza un conjunto de cables denominados bus de E/S. Existen varios tipos, como SATA o USB. Las transferencias de datos en bus son realizadas por las controladoras. La controladora host, es la controladora situada en el extremo del bus correspondiente a la computadora; además, en cada unidad de disco se integra una controladora de disco.

Cintas magnéticas. Pueden albergar grandes cantidades de datos, su tiempo de acceso es lento comparado con las memorias principales y las unidades de disco. Además, el acceso aleatorio es 1000 veces más lento que el de un disco. No resultan muy útiles como almacenamiento secundario, se utilizan para copias de seguridad o para almacenamiento de datos de uso poco frecuente.

12.2 Estructura de un disco

Las unidades de disco modernas se direccionan como grandes matrices unidimensionales de **bloques lógicos**, en las que el bloque lógico es la unidad más pequeña de transferencia. La matriz unidimensional de bloques lógicos se mapea sobre los sectores del disco secuencialmente. El sector 0 es el primer sector de la primera pista del cilindro más externo y el mapeo continua por orden a lo largo de dicha pista, después a lo largo del resto de las pistas de dicho cilindro y luego a lo largo del resto de los cilindros, desde el más externo al más interno.

PLANIFICACIÓN DE DISCO

Para un sistema de multiprogramación con múltiples procesos, la cola de disco puede terne varias solicitudes pendientes. Cuando se completa una solicitud, el sistema operativo selecciona cual es la próxima solicitud a atender, esto lo hace mediante algoritmos de planificación.

- Planificación FCFS. First come, first serverd; el primero el llegar el primero en ser servido.
- Planificación SSTF. Shorted seek time first. El algoritmo SSTF selecciona la solicitud que tenga el tiempo de búsqueda más corto con respecto a la posición actual del cabezal.
- Planificación SCAN. Comenzando por un extremo, el cabezal se mueve de un extremo a otro, dando servicio a las solicitudes a medida que pasa por cada cilindro, hasta llegar al otro extremo del disco.
- Planificación C-SCAN. Igual que el SCAN pero no atiende solicitudes cuando vuelve luego de alcanzar el extremo final.
- Planificación LOOK. Sigue la lógica de SCAN y C-SCAN, pero antes de ir a un extremo miran si hay una solicitud antes de continuar moviéndose. Es decir los extremos lo determinan las solicitudes más alejadas.

12.3 Gestión de disco

FORMATEO DE DISCO

Antes de poder almacenar datos en el disco, es necesario dividir este en sectores que la controladora de disco pueda leer y escribir. Este proceso se denomina formateo de bajo nivel o formateo físico. Esté llena el disco con una estructura de datos especial para cada sector.

Esta estructura consta típicamente de una cabecera, un área de datos y una cola. La cabecera y la cola contienen información utilizada por la controladora de disco, como el número de sector y un código de corrección de errores (ECC). Se escribe un EEC al almacenar un conjunto de datos y se calcula otro cuando se lee el sector. Si el ECC almacenado y el calculado no coinciden, indica que el área de datos ha sido corrompida y el sector de datos puede ser defectuoso. Si el error en los datos es de unos pocos bits se puede corregir y se lo llama un error blando.

Para utilizar un disco para almacenar archivos, debemos seguir dos pasos:

- Primero debemos particionar el disco en uno o más grupos de cilindros. El SO puede tratar una partición como si fuese un disco distinto.
- Luego debemos realizar el formateo lógico, donde el SO almacena las estructuras de datos iniciales del sistema de archivos en el disco.

La mayoría de los sistemas de archivos agrupan los bloques en una serie de fragmentos de mayor tamaño, denominados clusters.

Bloque de arranque

Para que una computadora comience a operar debe tener un programa inicial que ejecutar. Este programa inicial de arranque tiende a ser muy simple. La mayoría de los sistemas almacenan un programa cargador de muy pequeño tamaño en la ROM (read only memory) de arranque, cuya única tarea consiste en cargar un programa de arranque completo desde el disco. De esta forma el programa de arranque puede cambiarse fácilmente. Un programa cargador completo se almacena en los bloques de arranque en una ubicación fija en el disco. Un disco que tenga una partición de arranque se denomina disco de arranque o disco del sistema.

RAID

RAID (conjunto redundante de discos Independiente), hace referencia a un sistema de almacenamiento de datos en tiempo real que utiliza múltiples unidades de almacenamiento de datos (discos) entre los que se distribuyen o replican los datos.

Dependiendo de su configuración (nivel), los beneficios de un RAID respecto a un único disco son uno o varios de los siguientes: mayor integridad, mayor tolerancia a fallos, mayor rendimiento y mayor capacidad.

Niveles

- **RAID 0:** distribuye los datos equitativamente (a nivel de bloques) entre dos o más discos sin proporcionar redundancia alguna.
- **RAID 1:** crea una copia exacta (o espejo) de un conjunto de datos en dos o más discos. Esto resulta útil cuando queremos tener más seguridad desaprovechando capacidad, ya que si perdemos un disco, tenemos el otro con la misma información.
- **RAID 2:** usa división a nivel de bits con un disco de paridad dedicado y usa un código para la corrección de errores (ECC). Este tipo de RAID, adapta el mecanismo de detección de fallas en discos rígidos para funcionar en memoria. Así, todos los discos de la matriz están siendo "monitorizados" por el mecanismo.
- **RAID 3:** En este nivel, los datos son divididos entre los discos de la matriz, excepto uno, que almacena información de paridad. Así, todos los bytes de los datos tienen su paridad (aumento de 1 bit, que permite identificar errores) almacenada en un disco específico. A través de la verificación de esta información, es posible asegurar la integridad de los datos.
- **RAID 4:** Este tipo de RAID, básicamente, divide los datos entre los discos, siendo uno de esos discos exclusivo para paridad. La diferencia entre el nivel 4 y el nivel 3, es que en caso de falla de uno de los discos, los datos pueden ser reconstruidos en tiempo real a través de la utilización de la paridad calculada a partir de los otros discos, siendo que cada uno puede ser accedido de forma independiente.

- **RAID 5:** Este es muy semejante al nivel 4, excepto por el hecho de que la paridad no está destinada a un único disco, sino a toda la matriz. Eso hace que la grabación de datos sea más rápida, pues no es necesario acceder a un disco de paridad en cada grabación.
- **RAID 6:** amplía el nivel RAID 5 añadiendo otro bloque de paridad, por lo que divide los datos a nivel de bloques y distribuye los dos bloques de paridad entre todos los miembros del conjunto. Esto ayuda a la hora de múltiples fallos de disco.
- **RAID 0+1:** es una combinación de los niveles 0 (Striping) y 1 (Mirroring), donde los datos son divididos entre los discos para mejorar el ingreso, pero también utilizan otros discos para duplicar la información. Así, es posible utilizar el buen ingreso del nivel 0 con la redundancia del nivel 1. Sin embargo, es necesario por lo menos 4 discos para montar un RAID de este tipo. Estas características hacen del RAID 0 + 1 el más rápido y seguro, sin embargo es el más caro de ser implementado.
- **RAID 1+0:** a veces llamado RAID 10, es parecido a un RAID 0+1 con la excepción de que los niveles RAID que lo forman se invierte: el RAID 10 es una división de espejos.