Indice General

1− INTRODUCCIÓN HISTÓRICA.

Tras los problemas presentados por el sistema operativo OS/2, en el año 1988 Microsoft encargó a David N. Cutler (Dave), antiguo consultor senior en Digital Equipement Corporation, liderar el proyecto de creación de un nuevo sistema operativo de nueva tecnología (NT, New Technology) para los noventa. Después de casi un año de trabajo, Dave Cutler y su grupo de trabajo, identificaron como básicos los siguientes requisitos:

- **Portabilidad.** Los avances hardware suceden rápidamente y en muchos casos de forma impredicible. Escribir Windows NT en un lenguaje fácilmente transportable permitirá moverse libremente de una arquitectura de procesador a otra.
- **Multiproceso y facilidad de ampliación.** Las aplicaciones deberían ser capaces de aprovechar la amplia gama de ordenares disponibles actualmente. Con la aparición de ordenadores con mas de un procesador, hacer al NT escalable y con multiproceso, posibilitaría que un usuario ejecutase la misma aplicación en ordenadores mono o multiprocesador.
- **Procesamiento distribuido.** Windows NT, debido a las posibilidades avanzadas de trabajo en red, estaría dotado de herramientas de trabajo en red, y además, estaría capacitado para distribuir su trabajo a través de múltiples sistemas de ordenador.
- **Conformidad POSIX.** A mediados y finales de los años ochenta, las agencias de gobierno de EE.UU., comenzaron a especificar el POSIX como un estándar para los contratos informáticos con el gobierno. Este estándar, promueve que los fabricantes de interfaces de estilo UNIX hagan sus productos compatibles para que los programadores puedan mover fácilmente sus aplicaciones de un sistema a otro.
- **Certificado de seguridad oficial.** Además de la conformidad POSIX, el gobierno de los EE.UU., también estable unas pautas de seguridad informática para los diferentes tipos de aplicaciones gubernamentales. Lograr un ratio de seguridad aprobado por el gobierno, permite al sistema operativo competir contra otros en ese terreno. Por supuesto, muchos de estos requisitos necesarios son características beneficiosas para cualquier sistema multiusuario. Las pautas de seguridad especifican características como la protección de los recursos de un usuario frente a la intromisión de otro usuario, y establecen cuotas de recursos para evitar que un usuario acumule todos los recursos del sistema (como la memoria). El objetivo de seguridad inicial de Windows NT es el llamado nivel Clase C2, definido por el departamento de defensa de EE.UU., como protección discrecional y, con herramientas de auditoría para poder contabilizar los sujetos y las acciones que ellos inician. Esto significa que el propietario de un recurso tiene derecho quien puede acceder a él, y el sistema operativo puede detectar cuando y quien accede a los datos. Los niveles de seguridad de los EE.UU., van desde el nivel A (más riguroso) hasta el nivel D (menos riguroso), pasando por B y C, cada uno de los cuales tienen diversos subniveles. Aunque Windows NT fue escrito inicialmente para soportar el nivel de seguridad C2, las versiones futuras podrán ajustarse a niveles de seguridad mayores.

Con estos requisitos de mercado, el grupo de desarrollo de Windows NT tuvo una misión: La creación del sistema operativo de Microsoft para la década de los noventa.

- **OBJETIVOS DE DISEÑO.**

El diseño de Windows NT necesitó de profundas reflexiones. Para que el sistema pudiera cumplir los requisitos preestablecidos del mercado, era crucial que los rasgos complejos como la conformidad POSIX y la seguridad, fueran incorporados desde el comienzo.

Antes de la codificación, los diseñadores del sistema configuraron cuidadosamente un conjunto de objetivos de diseño. Estos facilitaron la toma de miles de decisiones que determinaron la estructura interna de un proyecto de tan grande envergadura. Las metas u objetivos siguientes son los objetivos de diseño de Windows NT:

- **Extensibilidad.** El código tiene que estar escrito para crecer cómodamente y cambiar según evolucionen las necesidades del mercado.
- **Facilidad de transporte.** Según estén dictadas las metas del mercado, el código debe poder moverse fácilmente de un procesador a otro.
- **Fiabilidad y robustez.** El sistema debería protegerse a sí mismo del mal funcionamiento interno y de los fallos externos. Debería comportarse de forma predecible en todo momento, y las aplicaciones no deberían dañar al sistema operativo o a su funcionamiento.
- **Compatibilidad.** Aunque Windows NT debería extender la tecnología existente, su interfaz de usuario y los API (application programming interface, o interfaz de programación de aplicaciones) deberían ser compatibles con los sistemas ya existentes en Microsoft.
- **Rendimiento.** Dentro de las limitaciones impuestas por el resto de los objetivos de diseño, el sistema debería ser lo más rápido posible en cualquier plataforma hardware.

En los siguientes apartados veremos mas a fondo los objetivos de diseño de Windows NT, y los efectos que éstos han tenido en la forma final del sistema operativo.

2.1− EXTENSIBILIDAD.

Invariablemente, los sistemas operativos cambian a lo largo del tiempo. Los cambios se presentan a menudo en forma de nuevas características.

Asegurar la integridad del código de Windows NT respecto a los cambios del sistema operativo, era un objetivo primordial de diseño. Para esto Windows NT comprende un ejecutor privilegiado y un conjunto de servidores no privilegiados llamados subsistemas protegidos. El término privilegiado hace referencia a los modos de funcionamiento del procesador. La mayoría de los procesadores poseen un modo de funcionamiento privilegiado (o quizás varios), en el cual el programa un programa puede emplear todas las instrucciones de la máquina y puede acceder a la memoria del sistema, y un modo no privilegiado en el que ciertas instrucciones no están permitidas y no puede acceder a la memoria del sistema. En Windows NT, el modo privilegiado se denomina modo kernel, y el modo no privilegiado se denomina modo usuario (user mode).

Generalmente, un sistema operativo se ejecuta sólo en modo kernel y los programas de aplicación se ejecutan sólo en modo usuario, excepto cuando solicitan servicios del sistema operativo. Sin embargo, el diseño de Windows NT es único, porque sus subsistemas protegidos se ejecutan en modo usuario al igual que las aplicaciones. Esta estructura permite que los subsistemas protegidos puedan ser modificados o que se añadan nuevos subsistemas sin afectar a la integridad del ejecutor.

Además de los subsistemas protegidos, Windows NT incluye numerosas características para asegurar su extensibilidad:

- **Estructura modular.** El ejecutor consta de un juego discreto de componentes individuales que interaccionan entre ellos solamente a través de interfaces funcionales. Pueden añadirse nuevos componentes al ejecutor de forma modular, ejecutándose su función al llamar a las interfaces suministradas por los componentes existentes.
- **Utilización de objetos para representar los recursos del sistema.** Los objetos, tipo de datos abstractos que son manipulados solamente por un conjunto especial de servicios, permiten que los recursos del sistema sean manejados uniformemente. La adición de nuevos objetos no altera objetos ya existentes ni requiere el cambio del código existente.

- **Drivers cargables.** El sistema de I/O de Windows NT soporta drivers que pueden ir añadiéndose al sistema a medida que se ejecuta. Se pueden soportar nuevos sistemas de ficheros, dispositivos, y redes, escribiendo drivers de dispositivo, de sistemas de ficheros, o de transporte, y cargándolos en el sistema.
- **Mecanismo de llamada de procedimiento remoto** (RPC, Remote procedure call), que permite que una aplicación llame a servicios remotos sin importar su situación dentro de la red. Se pueden añadir nuevos servicios a cualquier máquina de la red, y pueden ponerse inmediatamente a disposición del resto de los integrantes de la red.

2.2− TRANSPORTABILIDAD.

Este segundo objetivo, está estrechamente relacionado con la extensibilidad. La extensibilidad le permite a un sistema operativo ser fácilmente mejorado, y la portabilidad le permite a todo sistema operativo ser trasladado a otra máquina basada en otro microprocesador o configuración, con la menor modificación posible del código.

Windows NT fue escrito para poder transportarse fácilmente a máquinas que empleasen direcciones lineales de 32 bits y proporcionasen características de manejo de memoria virtual. También puede moverse a otro tipo de máquinas, pero con un coste mayor.

2.3− FIABILIDAD.

La fiabilidad se refiere a dos ideas diferentes pero relacionadas. Primero, un sistema operativo debería ser robusto, respondiendo de forma predecible a las condiciones de error, incluso ante aquellas originadas por fallos de hardware. Segundo, el sistema operativo debería protegerse activamente a sí mismo y a sus usuarios, de un sabotaje accidental o deliberado de los programas de usuario.

El manejo estructurado de excepciones es un método para capturar condiciones de error y responder a ellas uniformemente. Esta es la defensa principal de Windows NT frente a fallos en el software o el hardware. Siempre que sucede un evento anormal, o el sistema operativo o el procesador generan una excepción; el código de manejo de la excepción, existe en todo el sistema, se invoca automáticamente para responder a la condición, asegurando que ningún error no detectado haga estragos en los programas de usuario o en el mismo sistema.

2.4− COMPATIBILIDAD.

Es el cuarto objetivo de diseño de Windows NT, es un sistema complicado. En general, la compatibilidad se refiere a la habilidad de un sistema operativo de ejecutar código escrito para otro sistema operativo, o para versiones antiguas de es sistema operativo. Para Windows NT, el tema de la compatibilidad toma diversas formas.

Definir esta cuestión es un asunto de compatibilidad binaria versus compatibilidad de aplicaciones a nivel de fuentes. Se consigue compatibilidad binaria cuando se puede tomar un programa ejecutable y ejecutarlo con éxito en un sistema operativo diferente. La compatibilidad a nivel de fuentes implica recompilar el programa antes de ejecutarlo en el nuevo sistema.

El que el nuevo sistema operativo sea compatible en código binario o en código fuente con otro sistema existente, depende de varios factores. El primero de ellos es la arquitectura del nuevo procesador del sistema. Si el procesador emplea el mismo juego de instrucciones y las direcciones de memoria son del mismo tamaño que el antiguo, puede lograrse la compatibilidad binaria.

La compatibilidad binaria no es tan sencilla entre procesadores basados en arquitecturas diferentes. Cada

arquitectura de procesador por lo general conlleva un único lenguaje máquina. Esto significa que la arquitectura cruzada y la compatibilidad binaria solo pueden lograrse si se dispone de un programa emulador para convertir un conjunto de instrucciones de código máquina en otro.

Windows NT admite sistemas de ficheros existentes, incluyendo el sistema de ficheros MS−DOS (FAT), el sistema de ficheros de elevadas prestaciones del OS/2 (HPFS, High−performance file system), el sistema de ficheros de DD−ROM (CDFS, CD−ROM file system), y el nuevo y recuperable sistema de ficheros de Windows NT (NTFS).

2.5− RENDIMIENTO.

Los siguientes procesos ayudaron a lograr este objetivo:

- Cada componente de Windows NT fue diseñado teniendo en cuenta el rendimiento. La comprobación del rendimiento y la modelización se realizaron en las partes críticas del sistema. Las llamadas al sistema, los fallos de página, y otros caminos cruciales , se optimizaron cuidadosamente para asegurar la velocidad de procesamiento más rápida posible.
- Los subsistemas protegidos (servidores) que llevan a cabo funciones del sistema operativo se tienen que comunicar frecuentemente entre ellos y con aplicaciones cliente. Para garantizar que estas aplicaciones no entorpezcan el rendimiento del servidor, se incorporó un mecanismo de paso de mensajes de alta velocidad denominado llamada de procedimiento local (LPC, Local procedure call), como parte integral del sistema operativo.
- Cada subsistema protegido que proporcina un entorno del sistema operativo (subsistema de entorno) fue cuidadosamente diseñado para maximizar la velocidad de los servicios del sistema utilizados con más frecuencia.
- Los componentes cruciales del software de red de Windows NT, fueron integrados en la parte privilegiada del sistema operativo para asegurar la mayor efectividad posible. Aunque estos componentes se encuentran integrados, pueden cargarse y retirarse del sistema de forma dinámica.

3− EL KERNEL DE WINDOWS NT .

3.1− INTRODUCCIÓN.

El objetivo predominante para Dave Cutler para el kernel de Windows NT era proporcionar una base de bajo nivel de primitivas y mecanismos de sistema operativo predecibles, bien definidas, que permitieran que los componentes de mas alto nivel del ejecutor de Windows NT hicieran lo que necesitaran. Utilizando las primitivas del kernel, el ejecutor de Windows NT puede construir abstracciones de alto nivel de una variedad infinita. No necesitaba recurrir a interfaces de puerta trasera, a efectos laterales sin documentar, o a manipulación directa del hardware. El kernel está separado del resto del ejecutor, implementando los mecanismos de sistema operativo y evitando hacer cualquier tipo de políticas. Deja casi todas las decisiones políticas al ejecutor de Windows NT.

Al proporcionar un valioso conjunto de uniformes y controlados mecanismos, el kernel de Windows NT hace posible que Windows NT crezca y cambie con el tiempo.

3.2− VISION GENERAL.

Separar los mecanismos de sistema operativo de sus políticas es un principio importante en Windows NT. Los mecanismos se refieren a la forma en la cual se realizan las tareas en un sistema y están implementados por algoritmos y código. Las políticas determinan que tareas se deben de realizar y cuando, o incluso si deben realizarse ciertas tareas. El código de sistema operativo que mantiene una linea estricta entre los mecanismos y las políticas ayuda a que el sistema operativo se mantenga flexible. Las políticas pueden cambiar através del

tiempo sin que causen una oleada de cambios a través del sistema o fuercen a que cambien los mecanismos.

El principio de separar las políticas de los mecanismos existe en varios niveles en Windows NT. En el nivel más alto, cada subsistema de entorno estable una capa de políticas de sistema operativo que es distinta en cada subsistema. Bajo los subsistemas, el ejecutor de Windows NT establece otra capa más básica de políticas que se ajustan a todos los subsistemas. En la capa más baja del sistema operativo, el kernel evita las políticas enteramente. En su lugar, él mismo sirve como una capa entre el resto del sistema operativo y el procesador. Forzar a que todas las operaciones relacionadas con el procesador sean canalizadas através del kernel resulta en una gran portabilidad y predicibilidad. El ejecutor de Windows NT ejerce sólo un control limitado sobre estas operaciones al llamar a las funciones del kernel.

Además de las funciones que suministra al ejecutor de Windows NT, el kernel realiza cuatro tareas importantes:

- Planifica la ejecución de los hilos.
- Transfiere el control a las funciones del controlador cuando suceden interrupciones y excepciones.
- Lleva a cabo la sincronización de bajo nivel del multiprocesador.

- Implementa los procedimientos de recuperación del sistema después de un fallo de alimentación.

El kernel es diferente del resto del ejecutor en diversas formas. A diferencia de otras partes del ejecutor, el kernel nunca es paginado fuera de la memoria y la multitarea cesa durante cortos periodos de tiempo, mientras se ejecuta el kernel. El kernel siempre se ejecuta en modo kernel, el modo de procesador privilegiado de Windows NT, y está diseñado para ser pequeño, compacto y tan portable como el rendimiento y las diferencias entre las arquitecturas de los procesadores permitan.

Fuera del kernel, el ejecutor representa los hilos y otros recursos compartibles como objetos. Estos objetos requieren alguna sobrecarga de política, tal como handles de objeto para manipularlos, comprobaciones de seguridad para protegerlos, cuotas de recurso a ser deducidas cuando son creados, y los mecanismos de reserva y liberación de memoria para contenerlos. Esta sobrecarga es eliminada en el kernel, que implementa un conjunto de objetos más simples, denominados objetos del kernel, que ayudan al procesamiento central de control del kernel y soportan la creación de objetos del ejecutor. La mayoría de los objetos a nivel de ejecutor encierran uno o más objetos del kernel, incorporando así sus poderosos atributos definidos en el kernel.

Un conjunto de objetos del kernel, denominados objetos de control, establecen la semánticas para controlar las diversas funciones de sistema operativo. Este conjunto incluye el objeto proceso del kernel, el objeto llamada de procedimiento asíncrono (APC, asynchronous procedure call), el objeto llamada de procedimiento aplazado (DPC, deferred procedure call), y diversos objetos utilizados por el sistema de E/S, incluyendo el objeto interrupción, el objeto notificación de alimentación, y el objeto estado de alimentación. Otro conjunto de objetos del kernel, conocidos como objetos despachador, incorporan características de sincronización, y alteran o afectan la planificación de los hilos. Los objetos despachador incluyen el hilo del kernel, el mutex del kernel, el evento del kernel, el mutante del kernel, el par de eventos del kernel, el semáforo del kernel, y el timer del kernel. El ejecutor utiliza las funciones del kernel para crear instancias de los objetos del kernel, para manipularlos, y para crear los objetos más complejos que proporciona al modo usuario.

3.3− PLANIFICACIÓN Y DESPACHO DE HILOS.

Un hilo es una entidad ejecutable que se ejecuta en el espacio de direcciones de un proceso, utilizando los recursos reservados al proceso. Uno de los trabajos del kernel de Windows NT es llevar el seguimiento de los hilos que están listos para ejecutarse y seleccionar el orden en el que se ejecutarán, una tarea denominada planificación de hilos. Cuando las condiciones son correctas, el kernel selecciona un nuevo hilo para ejecutarse y realiza una selección de contexto para él. Una selección de contexto es el procedimineto de salvar

el estado máquina volátil asociado con la ejecución de un hilo, cargar el de otro hilo, y poner en marcha la ejecución del nuevo hilo. El módulo que realiza estas tareas es el despachador del kernel.

3.3.1− Procesos del kernel y objetos hilo.

El trabajo del despachador es asegurar que de todos los hilos que están esperando a ejecutarse, los procesadores estén siempre ejecutando los mas adecuados. Cuando suceden eventos del sistema que cambian el estado de algún hilo, el despachador examina la lista de hilos en espera y realiza una selección de contexto a un nuevo hilo si se requiere un cambio.

Aunque el despachador manipula los hilos, no comparte la misma vista de los hilos que tienen los programas en modo usuario o el resto del sistema operativo. El kernel trabaja con una versión reducida del objeto hilo, denominada objeto hilo del kernel. Este está contenido dentro de un objeto hilo del ejecutor y representa sólo la información que el kernel necesita para despachar la ejecución de un hilo. De forma similar, el kernel implementa una versión mínima del objeto proceso, denominada objeto proceso del kernel.

Ilustración 1.− Objeto proceso del kernel y objetos hilo del kernel.

La Ilustración 1 muestra las relaciones entre los objetos proceso e hilo del kernel, y sus análogos del ejecutor.

−−>[Author:T]

Ilustración 2.− Objeto proceso del kernel

Como muestra la Ilustración 2, el proceso del kernel contiene un puntero a una lista de hilos kernel. (El kernel no tiene conocimiento de handles, así que se salta la tabla de objetos). El proceso kernel también apunta al directorio de la tabla de páginas del proceso (utilizada para controlar el espacio de direcciones del proceso), al tiempo total que los hilos del proceso han estado ejecutándose, a la prioridad de planificación base por defecto del proceso, y al conjunto de procesadores por defecto en los que pueden ejecutarse los hilos (denominado afinidad con el procesador). El kernel mantiene control sobre la información almacenada en el objeto proceso del kernel. El resto del ejecutor puede leer o alterar la información solo llamando a una función del kernel. El objeto hilo del kernel es más complicado que el objeto proceso del kernel. Contiene alguna información obvia, tal como la afinidad con el procesador del hilo y la cantidad de tiempo total que el hilo ha empleado para ejecutarse. También incluye la prioridad de planificación base del hilo (que puede diferir de la prioridad de planificación base por defecto asignada a su proceso), y la prioridad actual del hilo. Una pequeña pero particularmente importante cantidad de datos, que contiene el objeto hilo del kernel es el estado del despachador de un hilo. Un hilo puede estar en uno de seis estados posibles en cualquier momento, sólo uno de los cuales hace al hilo susceptible de ser elegido para su ejecución.

El ciclo de vida de un hilo comienza cuando un programa crea un nuevo hilo. La petición llega hasta el ejecutor de Windows NT, donde el administrador de procesos reserva espacio para un objeto hilo y llama al kernel para inicializar el objeto hilo del kernel contenido dentro de él. Una vez inicializado, el hilo progresa a través de los siguientes estados:

- **Listo.** Cuando el despachador busca un hilo para que sea ejecutado, sólo considera los hilos que se encuentran en el estado listo. Estos hilos simplemente están esperando a ser ejecutados.
- **Standby.** Un hilo en estado standby ha sido seleccionado para su ejecución próxima en un procesador particular. Cuando se dan las condiciones correctas, el despachador realiza una selección de contexto para este hilo. Sólo un hilo puede estar en estado standby para cada procesador del sistema.
- **Ejecución.** Una vez que el despachador realiza una selección de contexto a un hilo, el hilo entra en el estado de ejecución, y se ejecuta. La ejecución del hilo continua hasta que, o el kernel lo desplaza para ejecutar un hilo con una prioridad más alta, o su cuantum de tiempo finaliza, o termina, o

voluntariamente entra en estado de espera.

- **Espera.** Un hilo puede entrar en el estado de espera de varias maneras: Un hilo puede esperar voluntariamente a un objeto para sincronizar su ejecución; el sistema operativo (el sistema de E/S por ejemplo) puede esperar en nombre del hilo; o un subsistema de entorno puede dirigir el hilo para suspenderse a sí mismo. Cuando finaliza la espera del hilo, el hilo vuelve al estado listo para volver a entrar en la planificación.
- **Transición.** Un hilo entra en el estado de transición si está listo para la ejecución pero los recursos que necesita no están disponibles. Cuando los recursos están disponibles, el hilo entra en el estado listo.
- **Terminado.** Cuando un hilo finaliza la ejecución, entra en el estado terminado. Una vez terminado, un objeto hilo podría ser o no eliminado.

3.3.2− Prioridades de planificación.

El despachador del kernel utiliza un esquema de prioridad para determinar el orden en el cual los hilos debería ejecutarse, planificando los hilos de prioridad más elevada antes que los que tienen las prioridades más bajas. El kernel incluso interrumpe o desplaza la ejecución de un hilo, si un hilo con prioridad más alta llega a estar listo para ejecutarse.

Inicialmente, un hilo obtiene su prioridad del proceso en el cual es creado. Por ejemplo , cuando un subsistema de entorno crea un proceso, asigna una prioridad base por defecto al proceso. El hilo hereda esa prioridad base y puede alterarla para que sea ligeramente más baja o más alta. Esta es la prioridad de planificación en la cual el hilo comienza su ejecución. La prioridad del hilo podría variar entonces desde esa base, cuando se ejecuta.

Para tomar decisiones en la planificación de hilos, el kernel mantiene un conjunto de estructuras de datos conocidas colectivamente como la base de datos del despachador. Esta base de datos controla qué hilos están esperando a ser ejecutados y qué procesadores están ejecutando qué hilos. La estructura de datos más importante en la base de datos del despachador se denomina cola de hilos listos del despachador. Esta cola es en realidad una serie de colas, una para cada prioridad de planificación.

El ejecutor de Windows NT soporta 32 niveles de prioridad, divididos en dos clases: tiempo real y prioridad variable. Los hilos de tiempo real, aquellos con prioridades desde 16 a 31, son hilos de alta prioridad utilizados por programas en los que el tiempo es un parámetro crítico.

Cuando el despachador vuelve a planificar un procesador, comienza con la cola de prioridad más alta, y la recorre hasta que encuentra un hilo; de este modo, planifica todos los hilos en tiempo real antes de planificar los hilos de prioridad variable. La mayoría de los hilos del sistema entran dentro de la clase de prioridad variable, con prioridades en el rango de 1 a 15. (La prioridad 0 está reservada para uso del sistema). Estos hilos se denominan de prioridad variable porque el despachador ajusta sus prioridades cuando se ejecutan para optimizar el tiempo de respuesta del sistema.

Cuando no está sucediendo nada en el sistema, el kernel suministra un hilo (por procesador) que siempre puede ser ejecutado. Estos hilos se denominan hilos ociosos, y el despachador los trata como si su prioridad estuviera por debajo de la de todos los demás hilos. Un hilo ocioso simplemente ejecuta un bucle, comprobando si otro hilo ha entrado en estado standby y está listo para ejecutarse en su procesador.

3.3.3− Selección de contexto.

Después de que un hilo se ejecute durante un cuantum de tiempo completo, el kernel lo desplaza y vuelve a planificar el procesador. Sin embargo, que expire el cuantum de tiempo no es lo único que inicia la planificación de hilos. También se pone en marcha la planificación cuando el hilo y su ejecución ya no es de

la prioridad más alta.

El propósito de volver a hacer la planificación es seleccionar el próximo hilo que se va a ejecutar en un procesador particular y colocarlo en el estado de espera. Al volver a planificar los hilos, el kernel utiliza la base de datos del despachador para determinar rápidamente qué procesadores están ocupados, cuáles están ociosos (ejecutando hilos ociosos) y qué prioridad de hilo está ejecutando cada procesador.

Un contexto de hilo y el procedimiento para seleccionar el contexto varía en función de la arquitectura del procesador. Una selección de contexto típica requiere salvar y volver a cargar los siguientes datos:

- Contador de programa.
- Registro de estado del procesador.
- Contenidos de otros registros.
- Punteros de pila del kernel y del usuario.
- Un puntero al espacio de direcciones en el cual se ejecuta el hilo (directorio de la tabla de páginas del proceso).

Para hacer una selección de contexto, el kernel salva esta información empujándola en la pila en modo kernel del hilo actual, y actualizando el puntero de pila. El kernel carga el contexto del nuevo hilo y, si el nuevo hilo está en un proceso diferente, carga la dirección de su directorio de tabla de páginas para que su espacio de direcciones esté disponible. Después de que el kernel haga alguna limpieza, el control pasa al contador de programa almacenado del nuevo hilo y este hilo comienza la ejecución.

3.4− MANEJO DE INTERRUPCIONES Y EXCEPCIONES.

Las interrupciones y las excepciones son condiciones del sistema operativo que desvían al procesador hacia código que está fuera del flujo normal de control. Pueden ser detectadas o por hardware o por software. Cuando se detecta una interrupción o una excepción, el procesador detiene lo que está haciendo, y transfiere el control a una localización especial de memoria, es decir, la dirección del código que gestiona la condición. En Windows NT, este código se denomina controlador del bloqueo.

El kernel de Windows NT distingue entre interrupciones y excepciones de la siguiente manera. Una interrupción es un evento asíncrono, que puede ocurrir en cualquier momento, no relacionado con lo que el procesador está ejecutando. Las interrupciones son generadas principalmente por los dispositivos de E/S, relojes de procesador o temporizadores, y pueden ser habilitadas o deshabilitadas.

Una excepción, en contraste, es una condición síncrona resultante de la ejecución de una instrucción particular. Las excepciones pueden ser reproducidas al ejecutar el mismo programa con los mismos datos bajo las mismas condiciones. Ejemplos de excepciones pueden ser violaciones de acceso a memoria, ciertas instrucciones del depurador, y errores de división por cero. El kernel de Windows NT también trata las llamadas a servicios del sistema como excepciones.

3.4.1− Controlador de bloqueo.

El término bloqueo se refiere al mecanismo de un procesador para capturar la ejecución de un hilo cuando sucede una excepción o una interrupción, cambiando de modo usuario a modo kernel, y transfiriendo el control a una localización fija del sistema operativo. En Windows NT el procesador transfiere el control al controlador de bloqueo del kernel del NT, un módulo que actúa como un panel de control, gestionando las excepciones y las interrupciones detectadas por el procesador y transfiriendo el control al código que controla la condición.

Al ser invocado, el controlador de bloqueo deshabilita brevemente las interrupciones mientras salva el estado

de la máquina. Crea un marco de bloqueo en el cual almacena el estado de la ejecución del hilo interrumpido. Esta información permite al kernel continuar la ejecución del hilo después de gestionar la interrupción o la excepción. El marco de bloqueo es normalmente un subconjunto del contexto completo de un hilo. El código de un hilo varía con la arquitectura del procesador.

El controlador de bloqueo resuelve algunos problemas por sí mismo, tales como algunas excepciones de direcciones virtuales, pero en la mayoría de los casos, determina la condición que ocurrió y transfiere el control a otro kernel o módulos del ejecutor.

3.4.2− Despacho de Interrupciones.

Las interrupciones generadas por hardware se originan normalmente en los dispositivos de E/S que deben notificar al procesador cuando necesiten un servicio. Los dispositivos activados mediante interrupción permiten que el sistema operativo consiga la máxima utilización del procesador al solapar el procesamiento central con las operaciones de E/S. El procesador pone en marcha una transferencia de E/S hacia o desde un dispositivo y después ejecuta otros hilos mientras el dispositivo completa la transferencia. Cuando el dispositivo termina, interrumpe al procesador para un servicio. Dispositivos generalmente activados por interrupción son: dispositivos apuntadores, impresoras, teclados, unidades de disco, y tarjetas de sonido.

El software del sistema operativo puede generar interrupciones. Por ejemplo, el kernel puede emitir una interrupción software para iniciar el despacho de un hilo e interrumpir asíncronamente la ejecución de un hilo. El kernel puede deshabilitar las interrupciones para que el procesador no las reciba, pero lo hace pocas veces.

Un submódulo del controlador de bloqueo del kernel, denominado despachador de interrupciones responde a las interrupciones. Determina la fuente de una interrupción y transfiere el control o a una rutina externa que controla la interrupción, denominada rutina de servicio de interrupción (ISR), o a una rutina interna del kernel que responde a la interrupción. Los drivers de dispositivo suministran ISR para servir las interrupciones de dispositivo, y el kernel proporciona rutinas de manejo de interrupciones para otros tipos de interrupciones.

3.4.2.1− Tipos de interrupciones y prioridades.

Diferentes procesadores son capaces de reconocer distintos números y tipos de interrupciones. El despachador de interrupciones mapea los niveles de interrupción hardware en un conjunto estándar de niveles de petición de interrupción (IRQL, interrupt request level) reconocidos por el sistema operativo.

Los IRQL ordenan las interrupciones por prioridad. Las prioridades de los IRQL son distintas de las prioridades de planificación descritas anteriormente. Una prioridad de planificación es una atributo de un hilo, mientras que un IRQL es un atributo de la fuente de una interrupción, como por ejemplo, un ratón o un teclado. Además, cada procesador tiene una configuración de IRQL que cambia cuando se ejecuta el código del sistema operativo. Un hilo ejecutándose en modo kernel puede aumentar o disminuir la configuración de IRQL del procesador en el cual se está ejecutando para enmascarar o desenmascarar las interrupciones de más bajo nivel.

El kernel define un conjunto de IRQL portables, que puede aumentar si un procesador tienen características especiales relacionadas con las interrupciones . Las interrupciones son servidas en orden de prioridad, y una de prioridad elevada, desplaza a una de prioridad inferior. En la Tabla 1se muestran los IRQL portables desde la prioridad más alta a la más baja.

Los IRQL, desde el nivel alto hasta el nivel 1 de dispositivo, están reservados para interrupciones hardware, y las interrupciones de nivel despacho/DPC y de nivel APC son interrupciones software que genera el kernel. El IRQL de nivel bajo no es realmente un nivel de interrupción; es la configuración en la que tiene lugar la ejecución normal de un hilo, y que permite que ocurran todas las interrupciones.

La configuración IRQL de cada procesador determina que interrupciones puede recibir ese procesador. Cuando se ejecuta un hilo en modo kernel, aumenta o disminuye el IRQL del procesador. Como muestra la Ilustración 3, las interrupciones de una fuente con un IRQL superior a la configuración actual interrumpen al procesador, mientras que las interrupciones de fuentes con los IRQL iguales o inferiores al nivel actual son bloqueadas, o enmascaradas, hasta que un hilo que se está ejecutando disminuye el IRQL.

Un hilo en modo kernel aumenta y disminuye el IRQL del procesador en el cual se está ejecutando, dependiendo de lo que esté intentando hacer. Por ejemplo cuando ocurre una interrupción, el controlador de bloqueo (ó quizá el procesador) eleva el IRQL del procesador hasta el IRQL, asignando a la fuente de interrupción. Esto bloquea todas las interrupciones iguales e inferiores a ese IRQL (sólo en ese procesador), lo cual asegura que el procesador que sirve a la interrupción no va a ser molestado por una interrupción menos importante. Las interrupciones enmascaradas son controladas por otro procesador, retenidas hasta que disminuya el IRQL. Cambiar el IRQL de un procesador es una operación muy potente que debe realizarse con mucho cuidado. Los hilos en modo usuario pueden cambiar el IRQL del procesador.

IRQL	**Tipo de interrupción**
Nivel alto	Comprobación de máquina o error de bus
Nivel de alimentación	Fallo de la alimentación
Nivel de la interrupción interprocesador	Solicitud de trabajo de otro procesador
Nivel de reloj	Intervalo de reloj
Nivel de dispositivo	Prioridad más elevada de un dispositivo de E/S
Nivel de dispositivo	Prioridad más baja de un dispositivo de E/S
Nivel despacho/DPC	Despacho de un hilo y procesamiento de una llamada de procedimiento aplazada (DPC)
Nivel APC	Procesamiento de una llamada de procedimiento asíncrona (APC)
Nivel bajo	Ejecución normal de un hilo

Tabla 1.− Niveles de petición de interrupción (IRQL).

Cada nivel de interrupción tiene un propósito específico. El procesador emite un nivel de interrupción alto cuando detecta una caída en el voltaje de la fuente de alimentación. Una interrupción de nivel de alimentación provoca que el sistema operativo se autodetenga, grabando la información crítica del estado del sistema antes de hacerlo, para que pueda volverse a poner en marcha cuando se restablezca la alimentación y continúe donde lo dejó. El kernel emite una interrupción interprocesador (IPI) para solicitar que otro procesador realice una acción, tal como despachar la ejecución de un hilo particular o actualizar su cache TLB. El reloj del sistema genera una interrupción a intervalos regulares, y el kernel responde actualizando el reloj y midiendo el tiempo de ejecución del hilo. Si un procesador soporta dos relojes, el kernel añade otro nivel de interrupción de reloj para realizar la medida. El kernel proporciona un número de niveles de interrupción para los dispositivos activados por interrupción; el número exacto varía con el procesador y la configuración del sistema. El kernel utiliza interrupciones software en el despacho DPC e IRQL APC para iniciar la planificación de hilos y para interrumpir asíncronamente la ejecución de un hilo.

Ilustración 3.− Enmascarado de interrupciones.

3.4.2.2− Procesamiento de interrupciones.

Cuando ocurre una interrupción, el controlador de bloqueo guarda el estado de la máquina y después llama al despachador de interrupciones. Este eleva inmediatamente el IRQL de procesador al nivel de la fuente de interrupción para enmascarar las interrupciones y baja ese nivel mientras progresa el servicio de la interrupción.

Al igual que muchos sistemas operativos, Windows NT utiliza una tabla de despacho de interrupción (IDT, interrupt dispatch table) para localizar la rutina que maneja una interrupción particular. El IRQL de la fuente de interrupción sirve como índice de la tabla, y las entradas de la tabla apuntan a las rutinas que manejan las interrupciones.

Después de ejecutarse la rutina de servicio, el despachador de interrupciones disminuye el IRQL del procesador hasta donde estaba antes de ocurrir la interrupción, y entonces vuelve a cargar el estado de la máquina salvado. El hilo interrumpido continúa ejecutándose donde estaba. Cuando el kernel disminuye el IRQL, podrían materializarse las interrupciones de prioridad inferior que estaban bloqueadas. Si esto sucede, el kernel repite el proceso para manejar la nueva interrupción.

Cada procesador tiene una tabla de despacho de interrupciones independiente para que si es necesario, diferentes procesadores puedan ejecutar distintas ISR.

La mayoría de las rutinas que manejan interrupciones residen en el kernel. El kernel actualiza el tiempo de reloj, por ejemplo, y cierra el sistema cuando sucede una interrupción de nivel de alimentación. Sin embargo, muchas interrupciones son generadas por dispositivos externos, como teclados, dispositivos apuntadores y unidades de disco. Por tanto los drivers de dispositivos necesitan una forma de indicar al kernel a que rutina debe llamar cuando tiene lugar una interrupción de dispositivo.

El kernel proporciona un mecanismo portátil (un objeto control del kernel denominado objeto interrupción) que permite a los drivers de dispositivo registrar las ISR para sus dispositivos. Un objeto interrupción contiene toda la información que el kernel necesita para asociar una ISR de dispositivo con un nivel de interrupción particular, incluyendo la dirección de la ISR, el IRQL en el cual interrumpe el dispositivo, y la entrada de la IDT del kernel con la que debería asociarse la ISR.

La asociación de una ISR con un nivel particular de interrupciones se denomina conexión de un objeto interrupción, y eliminar la asociación de una ISR de una entrada de la IDT se denomina desconexión de un objeto interrupción. Estas operaciones, realizadas llamando a una función del kernel, permiten que un driver de dispositivo active una ISR cuando dicho driver está cargado en el sistema y que la desconecte si el driver no está cargado.

La utilización del objeto interrupción para registrar una ISR evita que los drivers de dispositivo se relacionen directamente con el hardware de interrupción, que difiere entre las diversas arquitecturas de procesador, y que necesiten conocer detalles sobre la tabla de despacho de interrupciones. Esta característica del kernel ayuda a la creación de drivers de dispositivo portables, porque elimina la necesidad de codificar en lenguaje ensamblador, o de reflejar las diferencias de los procesadores en los drivers de dispositivo.

Los objetos interrupciones proporcionan otras ventajas. Al utilizar el objeto interrupción, el kernel puede sincronizar la ejecución de la ISR con otras partes de un driver de dispositivo que podría compartir datos con la ISR. Por tanto, los objetos interrupción permiten que el kernel llame fácilmente a más de una ISR para cada nivel de interrupción. Si múltiples drivers de dispositivo crean objetos interrupción y los conectan a la misma entrada de la IDT, el despachador de interrupción llama a cada rutina cuando sucede una interrupción en ese nivel. Esto permite que el kernel soporte fácilmente configuraciones daisy chain, en las cuales varios dispositivos provocan interrupciones con la misma prioridad.

3.4.2.3− Interrupciones software

Aunque el hardware genera la mayoría de la interrupciones, el kernel de Windows NT también genera interrupciones software para una variedad de tareas:

- Iniciación de despacho de hilos
- Manejo de la finalización de contadores
- Ejecución asíncrona de un procedimiento en el contexto de un hilo particular
- Soporte de operaciones asíncronas de E/S

Interrupciones de despacho. Cuando un hilo no puede continuar su ejecución por más tiempo, quizá porque ha terminado o porque espera un handle de objeto, el kernel llama directamente al despachador para efectuar una inmediata selección de contexto. Sin embargo, el kernel a veces detecta, cuando está inmerso en muchas capas de código, que debería ocurrir una nueva planificación. En esta situación, la solución ideal es solicitar el despacho pero aplazar la ocurrencia hasta que el kernel complete su actividad actual. Una manera conveniente de lograr esto es utilizar una interrupción software.

Con propósitos de sincronización, el kernel siempre eleva el IRQL del procesador hasta el nivel despacho/DPC o superior cuando se ejecuta, lo cual enmascara las interrupciones software (y desactiva el despacho de hilos). Cuando el kernel detecta que debería ocurrir el despacho, solicita una interrupción de nivel de despacho/DPC, pero debido a que el IRQL es igual o superior a ese nivel, disminuye el IRQL por debajo del nivel despacho/DPC, y tiene lugar la interrupción de despacho.

Interrupciones de llamada de procedimiento aplaza (DPC). Una manera de aplazar el despacho hasta que se den las condiciones correctas es activar el despacho utilizando una interrupción software. Windows NT utiliza las interrupciones software para aplazar también otros tipos de procesamiento.

El despacho tiene lugar en el IRQL de despacho/DPC. Las interrupciones que suceden en este nivel pasan através del controlador de bloqueo al despachador, el cual realiza la planificación de hilos. El kernel también procesa las llamadas de procedimiento aplazadas (DPC). Una DPC es una función que realiza una tarea del sistema, una tarea que es menos importante que la actual. Las funciones se denominan aplazadas porque no podrían ejecutarse inmediatamente. De manera similar a las interrupciones de despacho, las DPC se ejecutan sólo después de que el kernel (o, a menudo, el sistema de E/S) termina trabajos más importantes y disminuye el IRQL del procesador por debajo del nivel de despacho/DPC:

Las DPC proporcionan al sistema operativo la capacidad de generar una interrupción y ejecutar una función del sistema en modo kernel. El kernel utiliza las DPC para procesar la terminación del contador de tiempo (y liberar los hilos que están esperando dicho suceso) y volver a planificar el procesador tras expirar el cuantum de tiempo de un hilo. Los drivers de dispositivo utilizan las DPC para completar las peticiones de E/S.

Una DPC se representa por un objeto DPC, un objeto de control del kernel que no es visible para los programas de modo usuario pero si lo es para los drivers de dispositivo y otras partes del código del sistema. La parte más importante de información que contiene el objeto DPC es la dirección de la función del sistema que llamará al kernel cuando procese la interrupción DPC. Las rutinas DPC que están esperando a ejecutarse son almacenadas en una cola gestionada por el kernel denominada cola de DPC. Para solicitar una DPC, el código del sistema llama al kernel para inicializar un objeto DPC y después lo sitúa en una cola DPC.

Situar una DPC en la cola de DPC incita al kernel a solicitar una interrupción software en el nivel de despacho/DPC. Debido a que las DPC son situadas en la cola generalmente por un software que se está ejecutando en un nivel IRQL más alto, la interrupción solicitada no llega a la superficie hasta que el kernel baja el IRQL al nivel APC o nivel inferior.

Las rutinas de DPC se ejecutan "cubiertas"; es decir, cuando disminuye el IRQL se ejecutan sin considerar lo que el hilo está ejecutando. Debido a que los hilos en modo usuario se ejecutan en un IRQL bajo, hay buenas oportunidades de que una DPC interrumpa la ejecución de un hilo de usuario ordinario. Esto quiere decir, por ejemplo, que una DPC se podría ejecutar en el espacio de direcciones de un usuario con acceso a sus recursos sin que el usuario tenga conocimiento de ello. Debido a esto, y debido a que se ejecutan en el IRQL de despacho/DPC, las DPC no pueden adquirir recursos del sistema ni modificar la memoria virtual del hilo que toman prestado. Pueden llamar a funciones del kernel pero no pueden llamar a servicios del sistema, generar faltas de página, ni crear o esperar objetos.

Afortunadamente, sólo el código del sistema puede colocar una DPC en la cola, y el sistema operativo garantiza que sus DPC se comportan correctamente (los drivers de dispositivo deben asegurar también que las utilizan adecuadamente).

Las DPC son proporcionadas principalmente por los drivers de dispositivo, pero el kernel también las utiliza. El kernel utiliza una DPC sobre todo para controlar la terminación del cuantum de tiempo. En cada tick del reloj del sistema, sucede una interrupción en el IRQL de reloj. El controlador de la interrupción de reloj (ejecutándose en el IRQL de reloj) actualiza la hora del sistema y después decrementa un contador que controla cuanto tiempo lleva ejecutándose el hilo actual. Cuando el contador llega a cero, ha expirado el cuantum de tiempo del hilo, y el kernel podría necesitar volver a planificar el procesador, una tarea de prioridad inferior que debería hacerse en el IRQL de despacho/DPC. El controlador de interrupción del reloj coloca en la cola una DPC para iniciar el despacho de hilo y después termina su trabajo y disminuye el IRQL del procesador. Debido a que la interrupción de DPC tiene una prioridad más baja que las interrupciones de dispositivo, antes de que tenga lugar la interrupción de DPC se controlan las interrupciones de dispositivo pendientes que salgan a la superficie.

Interrupciones de llamada de procedimiento asíncrona (APC) Cuando el kernel sitúa un objeto DPC en la cola, la interrupción DPC resultante detiene la ejecución de cualquier hilo que se está ejecutando. A veces también es práctico poder interrumpir un hilo específico y hacer que ejecute un procedimiento.

El kernel proporciona los medios para hacerlo con lo que se denomina una llamada de procedimiento asíncrono (APC). El código del sistema y el código de modo usuario pueden situar en la cola una APC, aunque las APC de modo kernel son más potentes. Al igual que la DPC, una APC se ejecuta asíncronamente cuando las condiciones son apropiadas. Para las APC de modo usuario las condiciones impuestas son las siguientes:

- El hilo que va a ejecutar la APC debe estar ejecutándose.
- El IRQL del procesador debe estar en un nivel bajo.
- El hilo objetivo de la APC de modo usuario debe estar declarado como susceptible de ser alertado (un tema que se examinará en breve).

Las APC de modo kernel, a diferencia de las de modo usuario, no quieren permiso del hilo objetivo para ejecutarse en el contexto de ese hilo. Las APC de modo kernel pueden interrumpir un hilo y ejecutar un procedimiento sin la intervención o consentimiento del hilo.

Un programa sitúa en la cola una APC a un hilo particular llamando al kernel, o directamente (mediante el código del sistema), o indirectamente (mediante el código de modo usuario). El kernel, por su parte, solicita una interrupción software en el nivel APC, y cuando se dan todas las condiciones enumeradas arriba, el hilo objetivo es interrumpido y ejecuta la APC.

−−>[Author:T]**Ilustración 4.− Despacho de una excepción.**

Al igual que las DPC, las APC son descritas por un objeto control del kernel, denominado objeto APC. Las

APC que están esperando a ejecutarse residen en una cola de APC gestionada por el kernel. A diferencia de la cola de DPC, que pertenece a todo el sistema, la cola APC es específica de un hilo (cada hilo tiene su propia cola APC). Cuando se solicita situar en la cola una APC, el kernel inserta dicha APC en la cola perteneciente al hilo que ejecutar la rutina de la APC.

Aunque el código de modo usuario no puede crear o situar en la cola un objeto APC directamente, ciertos servicios nativos de Windows NT aceptan una rutina de APC de modo usuario como parámetro. Por ejemplo, un subsistema o una DL.L puede especificar una rutina de APC cuando establece un contador. Cuando el contador se agota, el kernel sitúa en la cola del subsistema la APC, y el subsistema la ejecuta. Si un subsistema pone una APC de Windows NT a disposición de las aplicaciones cliente, una aplicación podría, por ejemplo, utilizar una APC para recolectar basura a intervalos regulares. De manera similar, los servicios nativos de E/S toman una APC opcional como parámetro, que permite a un solicitante llevar a cabo una rutina basada en el resultado de una operación de E/S. (Aunque el subsistema Win32 no exporta directamente APC de Windows NT en sus API, proporciona capacidades de APC en sus funciones ReadFileExec( ) y WriteFileEx( )).

Un hilo, aunque no puede bloquear las APC de modo kernel, puede bloquear la distribución de las A PC de modo usuario. De hecho, un hilo debe indicar explícitamente su complacencia a aceptar una interrupción de APC de modo usuario declarándose a sí mismo como alertable. Esto puede hacerlo, o esperando un handle de objeto y especificando que su espera es alertable, o comprobando directamente si tiene una APC pendiente. En ambos casos, si hay una APC de modo usuario pendiente, el kernel interrumpe (alerta) al hilo, transfiere el control a la rutina de la APC, y continúa la ejecución del hilo cuando se completa la rutina de la APC.

El ejecutor de Windows NT utiliza las APC de modo kernel para llevar a cabo ciertas tareas de sistema operativo que deben completarse dentro del espacio de direcciones (del contexto) de un hilo particular. Puede utilizar las APC de modo kernel para hacer que un hilo deje de ejecutar un servicio del sistema ininterrumpible, por ejemplo, o para registrar el resultado de una operación de E/S asíncrona en el espacio de direcciones de un hilo. Los subsistemas de entorno utilizan las APC de modo kernel para hacer que un hilo suspenda, termine por si mismo, obtenga o establezca su contexto de ejecución de modo usuario.

3.4.3− Despacho de excepciones.

En contraste con las interrupciones, que pueden ocurrir en cualquier momento, las excepciones son condiciones que resultan directamente de la ejecución del programa que se está ejecutando.

Todas las excepciones, excepto aquellas suficientemente sencillas como para ser resueltas por el controlador de bloqueo, son atendidas por un módulo del kernel denominado despachador de excepción. La función del despachador de excepción es encontrar un controlador de excepción que pueda "deshacerse" de la excepción. A continuación se enumeran las excepciones no dependientes de la arquitectura, definidas por el kernel:

Violación de acceso a memoria. Desbordamiento de un entero

Punto flotante dividido por cero Breakpoint (punto de parada) del depurador

Instrucción ilegal Modo paso−a−paso del depurador

Error de lectura de página Entero dividido por cero

Desbordamiento de un punto flotante Operando reservado para punto flotante

Desalinemiento del tipo de datos Instrucción privilegiada

Vialación de página guardián Cuota de fichero de paginación excedida

El kernel de Windows NT atrapa y controla algunas de estas excepciones, sin que los programas de usuario tengan conocimiento de ello. Por ejemplo, encontrar un punto de parada del depurador mientras se ejecuta un programa que se está depurando, genera una excepción que el kernel maneja llamando al depurador. El kernel maneja otras excepciones al devolver un código de estado infructuoso al solicitante.

Unas cuantas excepciones pueden filtrarse de vuelta, intactas, al modo usuario. Por ejemplo, una violación de acceso a memoria, o un desbordamiento aritmético, genera una excepción no controlada por el sistema operativo. Un subsistema de entorno o una aplicación nativa pueden establecer controladores de excepción basados en marcos pana tratar estas excepciones, utilizando declaraciones de lenguaje de alto nivel diseñadas específicamente para el manejo de excepciones.

El término basados en marcos hace referencia a la asociación de un controlador de excepción con la activación de un procedimiento particular. Cuando se invoca un procedimiento, se empuja en la pila un marco de pila que representa esa activación del procedimiento. Un marco de pila puede tener uno o mas controladores de excepción asociados con él, cada uno de los cuales protege un bloque particular de código del programa fuente. Cuando tiene lugar una excepción, el kernel busca un controlador de excepción asociado con el marco de página actual. Si no existe ninguno, busca un controlador de excepción asociado con el marco de pila anterior, y así sucesivamente, hasta que encuentra un controlador de excepción basado en un marco. Si no encuentra ninguno, invoca a sus propios controladores de excepción por defecto. Obsérvese que el manejo de excepciones se implementa de distinta manera en los diferentes procesadores. La implementación del Intel x86 emplea un método de marcos de pila, mientras que la implementación del MIPS R4000 utiliza un método basado en tablas.

Cuando sucede una excepción, ya sea suscitada implícitamente por hardware o por software, se pone en marcha una cadena de eventos en el kernel. Se transfiere el control al controlador de bloqueo, que crea un marco de bloqueo (igual que cuando tiene lugar una interrupción). El marco de bloqueo permitir al sistema continuar donde lo dejó si se resuelve la excepción. También crea un registro de excepción que contiene la razón de ser de la misma y otra información al respecto.

Si la excepción ocurre en modo kernel, el despachador de excepción simplemente llama a una rutina para localizar un controlador de excepción basado en marco que maneje la excepción. Puesto que las excepciones de modo kernel no controladas son consideradas errores fatales del sistema, se puede asumir que el despachador siempre encuentra un controlador de excepción.

Si la excepción tiene lugar en modo usuario, el despachador de excepción lleva a cabo una tarea más elaborada. Un subsistema de entorno puede establecer un puerto de depuración y un puerto de excepción para cada proceso que crea. El kernel los utiliza en el control de excepciones por defecto, como muestra la Ilustración 4.

Los puntos de parada del depurador son fuentes muy comunes de excepciones. Por lo tanto, la primera acción que debe realizar el despachador de excepciones es enviar un mensaje (mediante la LPC) al puerto de depuración asociado con el proceso que emite la excepción. Esto permite al usuario manipular las estructuras de datos y emitir comandos de depuración.

Si no hay ningún puerto de depuración registrado o si el depurador no controla la excepción, el despachador de excepciones pasa a modo usuario y llama a una rutina para encontrar un controlador de excepciones basado en marco. Si no encuentra ninguno, o si ninguno controla la excepción, vuelve a modo kernel y llama de nuevo al depurador para permitir que el usuario siga con la depuración.

Si no se esta ejecutando el depurador y no se encuentran controladores basados en marco, el kernel envía un

mensaje al puerto de excepciones asociado con el proceso del hilo. Este puerto de excepciones, si existe, fue registrado por el subsistema de entorno que controla a este hilo. El puerto de excepciones proporcional subsistema de entorno, que presumiblemente está escuchando en el puerto, la oportunidad de traducir la excepción de Windows NT a una señal o excepción específica del entorno. Por ejemplo, cuando POSIX toma un mensaje procedente del kernel sobre que uno de sus hilos generó una excepción, el subsistema POSIX envía una señal de tipo POSIX al hilo que causó la excepción.

Aunque por defecto el subsistema POSIX asocia un puerto de excepción a cada uno de sus procesos, otros subsistemas podrían no proporcionar un puerto, o no llevar a cabo ninguna acción cuando el kernel les informa de una excepción no controlada en uno de sus procesos. Si el kernel sigue procesando la excepción y el subsistema no controla la excepción, el kernel ejecuta un controlador de excepción por defecto que sencillamente termina el proceso cuyo hilo originó la excepción.

3.4.4− Despacho de servicios del sistema.

El controlador de bloqueo de Windows NT despacha interrupciones, excepciones y llamadas de servicios del sistema. En la sección anterior se describía el manejo de las excepciones y las interrupciones, y esta sección examina brevemente los servicios del sistema. Las llamadas de servicios del sistema, las cuales generan bloqueos que son tratados como excepciones en Windows NT, son interesantes desde el punto de vista de la extensibilidad del sistema. La forma en la que el kernel implementa los servicios del sistema operativo permite que le sean añadidos dinámicamente nuevos servicios en versiones futuras.

Siempre que un hilo en modo usuario llama a un servicio del sistema, se le permite ejecutar inmediatamente código privilegiado del sistema operativo.

Ordinariamente esto es un anatema para el sistema operativo. Un hilo en modo usuario podría entrometerse en las estructuras de datos del sistema operativo o mover cosas por la memoria, causando estragos en el sistema o en otros usuarios. Por esta razón, los procesadores generalmente proporcionan una instrucción especial só1o para los sistemas operativos. Dicha instrucción (syscall en los procesadores MIPS R4000 e int 2Eh en los procesadores Intel x86) se genera cuando un hilo en modo usuario llama a un servicio del sistema. El hardware emite un bloqueo y conmuta la ejecución de modo usuario a modo kernel. Cuando esto sucede, el kernel copia los argumentos del que realiza la llamada, desde la pila de modo usuario del hilo a su pila de modo kernel (para que el usuario no pueda cambiar los argumentos, ya sea de buen o mal grado), y entonces ejecuta el servicio del sistema.

Como ilustra la Ilustración 5, el kernel utiliza una tabla de despacho de servicios del sistema para encontrar dichos servicios. Esta tabla es similar a la tabla de despacho de interrupciones descrita anteriormente, excepto en que cada entrada contiene un puntero a un servicio del sistema en lugar de un puntero a una rutina de manejo de interrupciones.

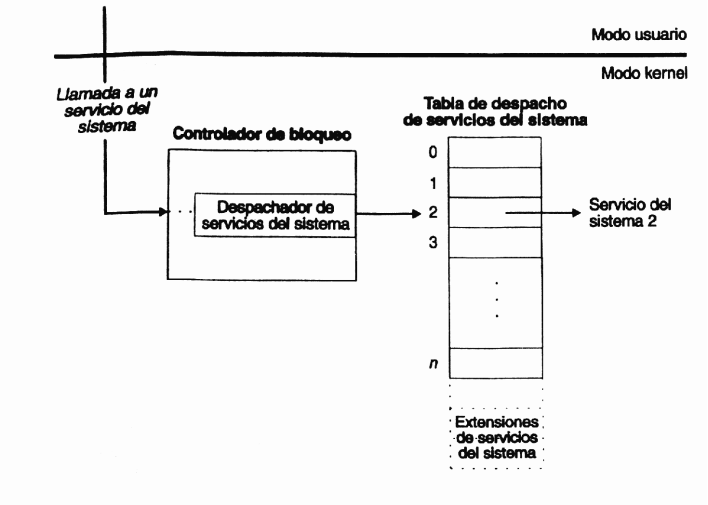
Utilizar una tabla de despacho de servicios del sistema proporciona la posibilidad de hacer extensibles los servicios nativos de Windows NT. El kernel puede soportar nuevos servicios simplemente ampliando la tabla, sin que sea necesario efectuar cambios en el sistema o en las aplicaciones. Cuando está escrito el código para un servicio nuevo, el administrador del sistema simplemente ejecuta un programa de utilidad que crea dinámicamente una nueva tabla de despacho. Esta nueva tabla contendría otra entrada que apuntarla al nuevo servicio del sistema. Aunque en la primera versión de Windows NT no está presente ni esta característica, ni su interfaz de usuario, pueden ser añadidas en un futuro.

3.5− SINCRONIZACIÓN DEL MULTIPROCESADOR.

El concepto de exclusión mutua es crucial en el desarrollo de los sistemas operativos. Se refiere a la garantía de que sólo un hilo pueda acceder a un recurso particular en un momento dado. La exclusión mutua es

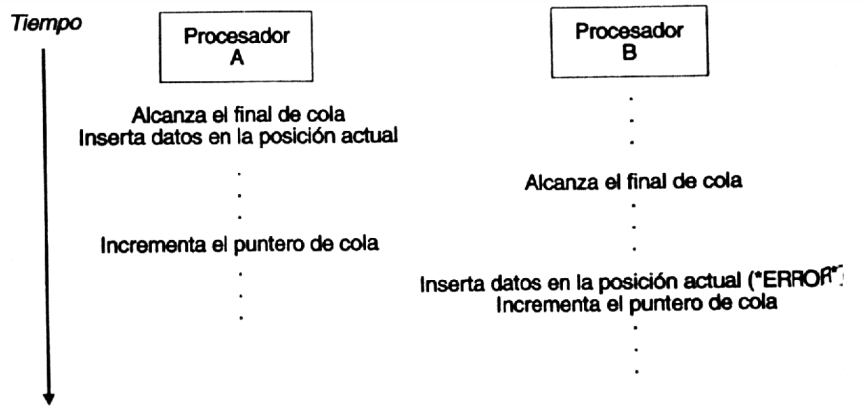
necesaria cuando un recurso no se presta a acceso compartido o cuando la compartición tendría un resultado impredecible. Por ejemplo, si dos hilos copian un fichero a un puerto de impresora a la vez, su salida podría ser entremezclada. De manera similar, si un hilo lee una localización de memoria mientras otro hilo escribe en ella, el primer hilo recibir datos impredecibles. En general, los recursos susceptibles de ser escritos no pueden compartirse sin restricciones, mientras que los recursos que no están sujetos a modificaciones si pueden ser compartidos. La Ilustración 6 ilustra lo que sucede cuando dos hilos que se ejecutan en diferentes procesadores escriben datos en una cola circular.

Puesto que el segundo hilo obtuvo el valor del puntero de final de cola antes de que el primer hilo lo hubiera actualizado, el segundo hilo insertó sus datos en la misma localización que había empleado el primer hilo, sobrescribiendo los datos y dejando vacía una localización de la cola. Obsérvese que aunque esta Ilustración ilustra lo que sucedería en un sistema multiprocesador, podría ocurrir el mismo error en un sistema con un só1o procesador si el sistema operativo se dispusiera a realizar una selección de contexto al segundo hilo antes de que el primer hilo hubiera actualizado el puntero de final de cola.



−−>[Author:T]**Ilustración 5.− Excepciones de servicios del sistema.**

Las secciones de código que acceden a recursos que no se pueden compartir se denominan secciones críticas. Para asegurar que el código en una sección crítica es correcto, sólo puede ejecutarse un hilo en un momento dado. Mientras que un hilo está escribiendo en un fichero, actualizando una base de datos, o modificando una variable compartida, a ningún otro hilo se le puede permitir acceder al mismo recurso. El código mostrado en la Ilustración 6 es una sección crítica que accede incorrectamente a una estructura de datos compartida sin exclusión mutua.



El asunto de la exclusión mutua, aunque importante en todos los sistema operativos, lo es especialmente en un sistema operativo con multiprocesamiento simétrico (SMP), estrechamente acoplado, como lo es Windows NT, en el cual el mismo código del sistema se ejecuta simultáneamente en más de un procesador, compartiendo ciertas estructuras de datos almacenadas en la memoria global.

Ilustración 6.− Compartición incorrecta de memoria.

En Windows NT, es tarea del kernel proporcionar mecanismos que el código del sistema pueda utilizar para prevenir que dos hilos modifiquen la misma estructura al mismo tiempo. El kernel proporciona primitivas de exclusión mutua que él y el resto del ejecutor utilizan para sincronizar sus accesos a estructuras de datos globales.

En la próxima subsección se describe cómo el kernel utiliza la exclusión mutua para proteger sus estructuras de datos globales. La sección siguiente se centra en los mecanismos de sincronización y exclusión mutua que proporciona el kernel al ejecutor y que él, a su vez, proporciona al modo usuario.

3.5.1− Sincronización del kernel.

El kernel debe garantizar, en varias etapas de su ejecución, que sólo un procesador está ejecutándose dentro de una sección crítica en un momento dado. Las secciones críticas del kernel son los segmentos de código que modifican una estructura de datos globales, tal como la base de datos del despachador del kernel o su cola de DPC. El sistema operativo no puede funcionar correctamente a menos que el kernel pueda garantizar que los hilos acceden a estas estructuras de una forma mutuamente excluyente.

El área de mayor preocupación son las interrupciones. Por ejemplo, el kernel podría estar actualizando una estructura de datos global en el momento en el que tiene lugar una interrupción cuya rutina de manejo de interrupción también modifica la estructura. Los sistema operativo de un sólo procesador previenen esa situación deshabilitando todas las interrupciones cada vez que acceden a datos globales, pero el kernel de Windows NT implementa una solución más sofisticada. Antes de utilizar un recurso global, el kernel enmascara temporalmente aquellas interrupciones cuyos controladores de interrupción también utilizan el recurso. Lo hace elevando el IRQL del procesador al nivel más alto utilizado por cualquier fuente de interrupción en potencia que pudiera acceder a los datos globales. Por ejemplo, una interrupción en el nivel de

despacho DPC hace que se ejecute el despachador, el cual utiliza la base de datos del despachador. Por lo tanto, cualquier otra parte del kernel que utilice la base de datos del despachador eleva el IRQL al nivel de despacho/DPC, enmascarando las interrupciones de nivel de despacho/DPC antes de utilizar la base de datos del despachador.

Esta estrategia es buena para un sistema con un sólo procesador, pero resulta inadecuada para una configuración de multiprocesador. Elevar el IRQL, en un procesador no evita que ocurra una interrupción en otro procesador. El kernel necesita garantizar también acceso mutuamente exclusivo através de los diversos procesadores.

El mecanismo que utiliza el kernel para lograr exclusión mutua en un sistema multiprocesador se denomina cierre de giro (spin lock). Se trata de un mecanismo de cierre asociado con una estructura de datos global, como por ejemplo, la cola de DPC.

Antes de entrar en la sección crítica, el kernel debe adquirir el cierre de giro asociado con la cola DPC protegida, Si el cierre de giro no está libre, el kernel continúa intentando su adquisición hasta que tenga éxito. Se denomina así al cierre de giro porque el kernel (y por lo tanto el procesador) se mantiene en el limbo, "girando y dando vueltas" hasta que consigue el cierre.

Los cierres de giro, al igual que las estructuras de datos que protegen, residen en la memoria global . El código para adquirir y liberar un cierre de giro está escrito en lenguaje ensamblador para acelerar y explotar cualquier mecanismo de cierre que proporcione la arquitectura subyacente del procesador. En muchas arquitecturas, los cierres de giro están implementados con una operación de prueba y colocación soportada por el hardware, que consiste en comprobar el valor de una variable de cierre y adquirir dicho cierre en una sola instrucción. Comprobar y adquirir el cierre en una instrucción evita que un segundo hilo se apodere del cierre en el intervalo de tiempo que transcurre desde que el primer hilo comprueba la variable hasta que adquiere el cierre.

Cuando un hilo está tratando de adquirir un cierre de giro, cesan el resto de las actividades en ese procesador. Por lo tanto, un hilo que sostiene un cierre de giro nunca es desplazado, sino que se le permite continuar ejecutándose para que libere el cierre de giro rápidamente. El kernel utiliza los cierres de giro con mucho cuidado, minimizando el número de instrucciones que ejecuta mientras lo sostiene.

El kernel pone los cierres de giro a disposición de otras partes del ejecutor a través de un conjunto de funciones del kernel. Los drivers de dispositivo, por ejemplo, necesitan cierres de giro para garantizar que sólo una parte del driver de un dispositivo accede a los registros de los dispositivos y a otras estructuras de datos globales en un momento dado.

3.5.2− Sincronización del ejecutor

El software del ejecutor fuera del kernel también necesita sincronizar el acceso a las estructuras de datos globales de un entorno multiprocesador. Por ejemplo, el administrador de VM tiene sólo una base de datos de marcos de página a la que se accede como una estructura de datos global, y los drivers de dispositivo necesitan asegurarse de que pueden lograr acceso exclusivo a sus dispositivos. Llamando a funciones del kernel, el ejecutor puede crear un cierre de giro adquirirlo, y liberarlo.

Sin embargo, los cierres de giro resuelven sólo parcialmente las necesidades del ejecutor en cuanto a mecanismos de sincronización. Puesto que esperar un cierre de giro literalmente para un procesador los cierres de giro sólo pueden utilizarse bajo las siguientes circunstancias estrictamente limitadas:

Cuando se debe acceder rápidamente al recurso protegido y sin interacciones complicadas con otro código.

Cuando el código de la sección crítica no puede ser paginado fuera de la memoria, no puede hacer referencias a datos paginables, no puede llamar a procedimientos externos (incluyendo los servicios del sistema), y no puede generar interrupciones ni excepciones.

Estas son restricciones límite, y no se dan en cualquier circunstancia. Además, el ejecutor necesita llevar a cabo otros tipos de sincronización aparte de la exclusión mutua y debe también proporcionar mecanismos de sincronización para el modo usuario.

El kernel suministra al ejecutor mecanismos de sincronización adicionales en forma de objetos del kernel, conocidos colectivamente como objetos del despachador. Un hilo puede sincronizarse con un objeto del despachador esperando un handle del objeto. Hacer esto provoca que el kernel suspenda el hilo y cambie su estado despachador desde el estado de ejecución al de espera, como resalta la Ilustración 7.

El kernel elimina el hilo de la cola de hilos listos del despachador y no vuelve a considerarlo para la ejecución. Un hilo no puede continuar su ejecución hasta que el kernel cambie su estado despachador desde el estado de espera al estado listo. Este cambio tiene lugar cuando el objeto despachador, cuyo handle está esperando el hilo, también sufra un cambio de estado, desde el estado no señalado al estado señalado (cuando un hilo establece un objeto evento, por ejemplo). El kernel es responsable de ambos tipos de transiciones.

Cada tipo de objeto despachador proporciona un tipo especializado de capacidad de sincronización. Por ejemplo, los objetos mutex proporcionan exclusión mutua, mientras que los semáforos actúan como una puerta a través de la cual pueden pasar un número variable de hilos (útil cuando están disponibles un cierto número de recursos idénticos). Los eventos pueden ser utilizados, o para anunciar que ha ocurrido alguna acción, o para implementar la exclusión mutua. El par de eventos es el medio que tiene el kernel de soportar la LPC rápida, que es la forma optimizada de paso de mensajes que utiliza el subsistema Win32. Los contadores llegan a cero cuando ha transcurrido una cierta cantidad de tiempo de reloj. Un hilo puede esperar a que otro hilo termine, lo cual es útil para sincronizar las actividades de dos hilos que están cooperando. Los objetos despachador del kernel juntos proporcionan al ejecutor una gran flexibilidad en la sincronización de su ejecución.

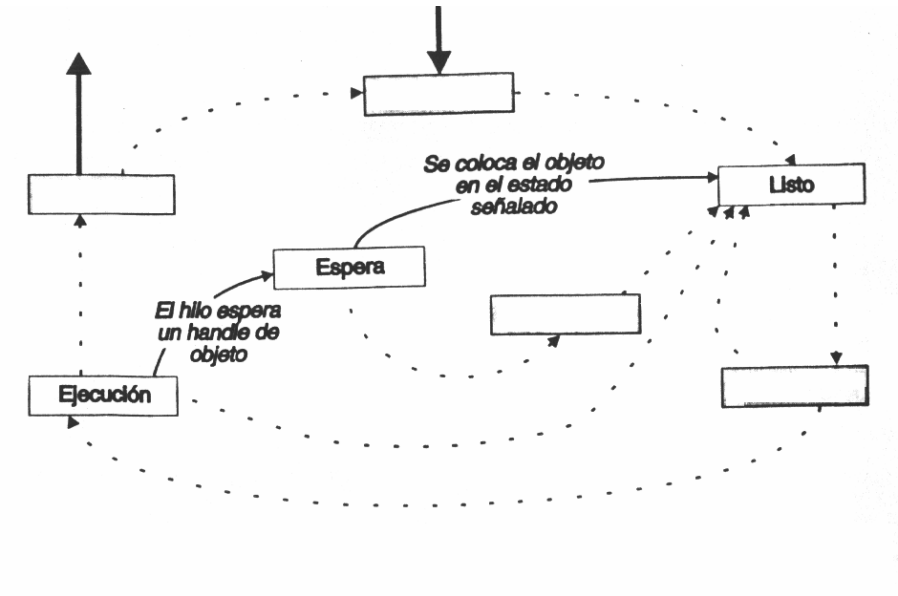


Ilustración 7.− Espera de un objeto despachador.

Los objetos sincronización visibles al usuario, adquieren sus capacidades de sincronización de los objetos despachador del kernel.

Cada objeto visible al usuario que soporta sincronización encierra al menos un objeto despachador del kernel. El siguiente ejemplo, sobre el establecimiento de un evento, ilustra cómo interacciona la sincronización con el despacho de hilos:

1. Un hilo de modo usuario espera un handle del objeto evento.

2 . El kernel cambia el estado de planificación del hilo desde el estado listo al estado de espera, y después añade el hilo a la lista de hilos que esperan el evento.

3. Otro hilo establece el evento.

4. El kernel recorre la lista de hilos que están esperando el evento. Si se satisfacen las condiciones de espera de un hilo, el kernel cambia el estado del hilo del estado de espera al estado listo. Si es un hilo de prioridad variable, el kernel podría elevar la prioridad de su ejecución.

5. Puesto que un hilo nuevo ha alcanzado el estado listo para ejecutarse, el despachador vuelve a planificar. Encuentra un hilo ejecutándose con una prioridad inferior de la del hilo nuevo que está listo, desplaza el hilo de prioridad inferior, y emite una interrupción software con el fin de iniciar una selección de contexto para el hilo de prioridad superior.

6. Si no puede ser desplazado ningún procesador, el despachador sitúa el hilo listo en la cola de hilos del despachador para que sea planificado más tarde.

4− EL SISTEMA DE ENTRADA / SALIDA.

4.1− INTRODUCCIÓN.

El sistema de entrada/salida (E/S) de Windows NT consta de diversos módulos de software, incluyendo el administrador de E/S y una serie de componentes que entran dentro de la clasificación de drivers. El administrador de E/S y una serie define el módelo de procesamiento de E/S de Windows NT, y realiza las funciones que son comunes o requeridas por más de un driver. Su principal responsabilidad es crear los IRP, que representan las peticiones de E/S, y guiar los paquetes a través de diversos drivers, devolviendo los resultados al solicitante cuando se completa la operación de E/S.

El término driver no incluye sólo a los drivers de dispositivo tracionales, sino también a los sistemas de ficheros, vías de comunicación con nombre, redes, y otros drivers. Todos los drivers adoptan una estructura común y se comunican entre ellos y con el administrador de E/S utilizando los mismos mecanismos. El administrador de E/S acepta las peticiones de E/S y localiza los diversos drivers utilizando objetos del sistema de E/S, entre los que se incluyen los objetos driver y los objetos dispositivo. Debido a que los drivers presentan una estructura común al sistema operativo, pueden estratificarse (colocarse en capas, unos sobre otros) para lograr una configuración modular y reducir la duplicación entre drivers. Internamente, el sistema de E/S funciona asíncronamente, ya que así consigue un alto rendimiento, y proporciona capacidades de E/S, tanto síncrona como asíncrona, a los subsistemas de modo usuario.

El diseño de drivers en Windows NT es diferente al diseño de drivers en otros sistemas operativos. Esto es debido a que los drivers deben trabajar correctamente en sistemas multiprocesador y poder participar en los procediminetos de recuperación ante fallos de alimentación de Windows NT. Sin embargo, todos los drivers están escritos en un lenguajes de alto nivel para reducir el tiempo de desarrollo y mejorar su portabilidad.

4.2− COMPONENTES DEL SISTEMA DE E/S.

La Ilustración 8 muestra una visión simplificada del sistema de E/S. El sistema de E/S está controlado por

paquetes, lo cual significa que cada petición de E/S se representa por un paquete de petición de E/S (IRP, I/O request packet), que viaja desde un componente del sistema de E/S a otro. Un IRP es una estructura de datos que controla cómo se procesa la operación de E/S en cada etapa del camino.

El administrador de E/S define un modelo en el cual las peticiones de E/S se entregan a los sistemas de ficheros y a los drivers de dispositivos. El administrador de E/S no gestiona realmente el procesamiento de la E/S. Su trabajo es crear un IRP que representa cada operación de E/S, pasar elIRP al driver correcto, y disponer como sea conveniente del paquete cuando se complete la operacion de E/S. En contraste, un driver recibe un IRP, realiza la operación que el IRP especifica, y , o lo devuelve al administrador de E/S, o lo pasa a otro driver (a través del administrador de E/S) para un procesamineto posterior.

Los sistemas de ficheros de Windows NT son drivers de dispositivo sofisticados que aceptan solicitudes de E/S a ficheros, y satisfacen dichas peticiones generandosus peticiones propias y más explícitas a los drivers de dispositivo físico. Los drivers del sistema de ficheros y los drivers de dispositivo se comunican pasando IRP.

Además de crear y disponer de los IRP, el administrador de E/S proporciona un código que es común a los diferentes drivers y que dichos drivers invocan para llevar a cabo sus procesamientos de E/S. Consolidando tareas comunes en el administrador de E/S, los drivers individuales resultan más sencillos y compactos.

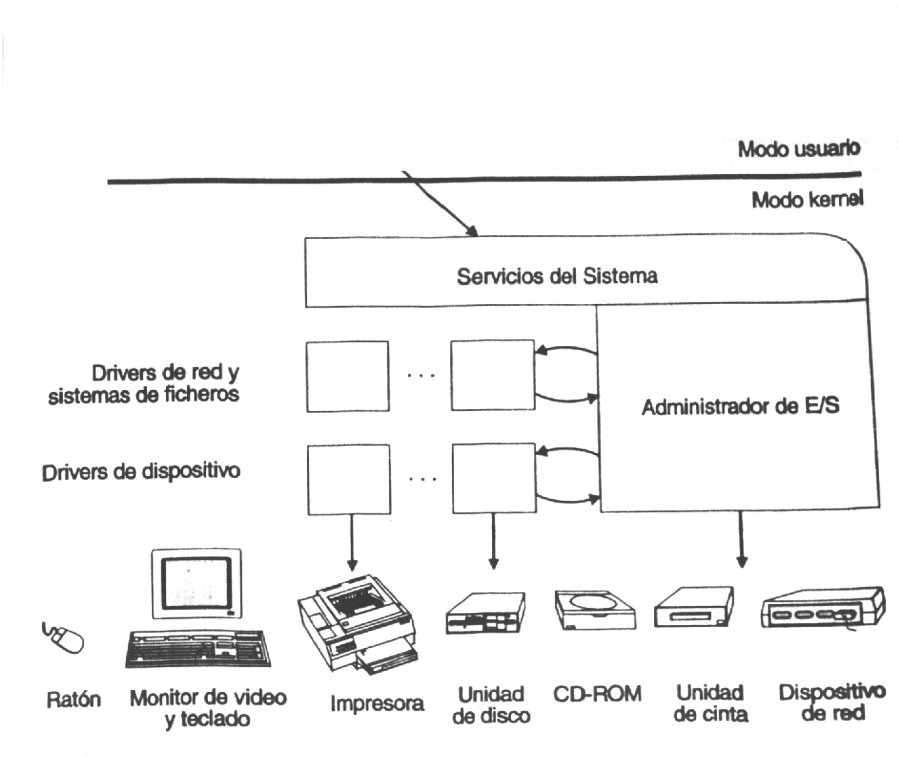


Ilustración 8.− Partes del sistema de E/S.

4.3− CARACTERÍSTICAS DE DISEÑO DE WINDOWS NT.

Algunas de las características de definición del modelo de E/S de Windows NT son:

- **Modelo de objetos de Windows NT.** Los programas también realizan operaciones de E/S en ficheros virtuales, manipulándolos mediante handles de ficheros. En Windows NT un handle de fichero se refiere a un objeto fichero del ejecutor. Los hilos de modo usuario llaman a servicios llaman a servicios nativos de los objetos fichero de Windows NT para leer de un fichero, escribir a un fichero,

y llevar a cabo otras operaciones. El administrador de E/S dirige dinámicamente estas solicitudes de ficheros virtuales a los ficheros reales, a los directorios de ficheros, a los dispositivos físicos, a las vías de comunicación (pipes), a las redes, a los buzones de correos (mailslots), y a cualquier otro destino que sea soportado en un futuro. Los orígenes y destinos de la E/S toman la forma de objetos por que se ajustan bien a los criterios de los objetos de Windows NT: son recursos del sistema que pueden ser compartidos por hilos de dos o más procesos de modo ususario. Los objetos fichero, como otros objetos, tienen nombres jerárquicos, están protegidos por la seguridad basada en los objetos, soportan síncronozación, y son manipulados por los servicios de objeto.

- **Modelo de driver uniforme.** El interfaz uniforme y modular que presentan los drivers permite que el administrador de E/S llame a cualquier driver a ciegas, sin requerir ningún conocimiento especial de su estructura o detalles internos. Existe la posibilidad de estratificar los drivers, permitiendo que éstos sean modulares e incrementando la capacidad de volver a utilizar el código del driver. Existen drivers de multiples capas y drivers de una sola capa, siendo más comunes los primeros aunque algunos dispositivos sencillos puedan utilizar los drivers de una sola capa.
- **Operación asíncrona.** El administrador de E/S de Windows NT proporciona la capacidad de llevar a cabo operaciones de E/S asíncronas. El mismo subsistema elige el empleo de E/S síncrona o asíncrona, y, dependiendo cómo funciona el API, puede proporcionar cada tipo de E/S a sus aplicaciones clientes. La E/S asíncrona proporciona un adelanto importante frente a la E/S síncrona: la mejora potencial de la velocidad de ejecución de una aplicación. Mientras el dispositivo está ocupado transfiriendo datos, la aplicación continúa con otro trabajo.
- **E/S mapeada en ficheros y cache de ficheros.** Las aplicaciones que realizan muchas operaciones de E/S con ficheros o que acceden a partes de muchos ficheros diferentes, pueden acelerar su ejecución utilizando la E/S mapeada, ya que escribir en memoria es mucho más rápido que escribir en un dispositivo. También el administrador de memoria virtual (VM) optimiza sus accesos a disco, así que la E/S mapeada permite que las aplicaciones se beneficien de esta característica. Un componente del sistema de E/S denominado administrador de cache utiliza E/S mapeada para gestionar su cache basado en memoria. Los sistemas de ficheros y el servidor de Windows NT utilizan la cache para situar en la memoria datos de ficheros a los que se accede confrecuencia, y proporcinar un tiempo de respuesta mejor en los programas que dependen mucho de operaciones de E/S. Mientras que la mayoría de los sitemas de cache reservan un número fijo de byhtes para la cache de ficheros en memoria, la cache de Windows NT aumenta o disminuye según la cantidad de memoria disponible.

4.4− OBJETOS FICHERO.

Un fichero es una representación basada en memoria de un recurso que se puede compartir, y no el recurso mismo, esta característica le hace diferente de otros objetos del ejecutor.

Un objeto fichero contiene datos que son exclusivos de un handle de objeto, mientras que el fichero en sí contiene datos o texto para ser compartidos. Cada vez que un hilo abre un handle de fichero, se crea un nuevo objeto fichero con un nuevo conjuntos de atributos específicos del handle.

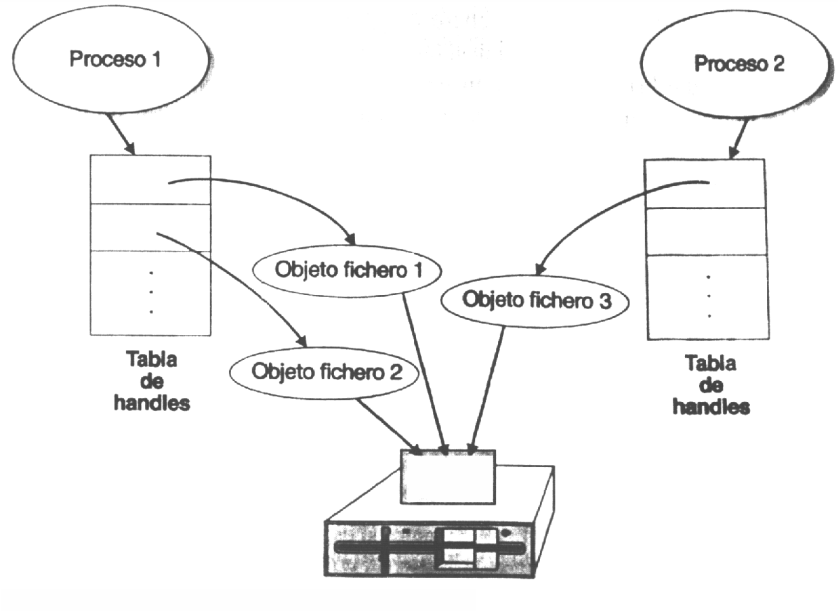


Ilustración 9.− Compartición de un fichero.

Aunque un handle de fichero es exclusivo de un proceso, el recurso físico subyacente no lo es. Por lo tanto, cuando utilizan algún recurso compartido, los hilos deben sincronizar sus accesos, ya sean ficheros que se puedan compartir, directorios de ficheros, o dispositivos.

1

Proceso del kernel

Token de acceso

Hilo del kernel

Hilo del kernel

Tabla de Objetos

Objetos hilo

Descriptores del espacio de direcciones virtuales (VAD)

Objeto proceso

Handle 1

Handle 2

Objeto hilo del kernel

Objeto hilo del kernel

Objeto proceso del kernel

Directorio de la tabla de página

Objeto hilo del kernel

Objeto hilo del kernel

Objeto hilo del kernel

Objeto proceso

Objetos hilo

Alto

Alimentación

Notificación interprocesador

Reloj

Dispositivo n

Dispositivo 1

Despacho/DPC

APC

Bajo

Interrupciones enmascaradas en al procesador A

Interrupciones enmascaradas en el procesador B

IRQL = Despacho/DPC

Procesador B

IRQL = Reloj

Procesador A

S.O.

(Ampl)

Subsistema de entorno o DLL

Fichero