

1. Fundamentos

Objetivos. En este tema se presentan los fundamentos sobre los que se va a trabajar en el resto de la asignatura. Es muy importante que estos conceptos queden muy claros antes de pasar a los siguientes temas. Estos conceptos básicos abarcan en primer lugar una serie de nociones sobre hardware y conceptos fundamentales sobre procesos, usuarios, etcétera, que probablemente sean conocidos por la mayoría de los alumnos, pero que en cualquier caso se recomienda leer detenidamente, pues cualquier error de concepto puede ser fatal a la hora de asimilar el resto de la materia. Y en segundo lugar, se exponen también las distintas estrategias de diseño que podemos elegir a la hora de construir un sistema operativo, mostrando a modo de ejemplo las arquitecturas de dos sistemas operativos que se utilizarán como casos de estudio en el resto de la asignatura. Dado que se hará referencia frecuentemente a aspectos de estos sistemas, es muy importante que su organización queda clara desde este tema.

1.1 Conceptos básicos

A continuación se exponen una serie de fundamentos que deben quedar muy claros antes de pasar a estudiar los siguientes temas. Estos fundamentos abarcan desde una serie de nociones básicas sobre arquitecturas de ordenadores y sobre procesadores, hasta una serie de conceptos relativos a sistemas operativos como son archivos, usuarios, procesos, conceptos que seguramente resultarán familiares. A pesar de ello, es importante que revise con atención los siguientes epígrafes, pues es posible que la idea intuitiva que pueda tener sobre algunos de estos conceptos no coincida al cien por cien con la noción que se emplea en esta asignatura.

1.1.1 Conceptos básicos de arquitectura de ordenadores

La arquitectura básica de un ordenador se compone de uno o más procesadores, interconectados con los circuitos de memoria y con los dispositivos de entrada y salida a través de un bus, tal y como se muestra en la Ilustración 1-1.

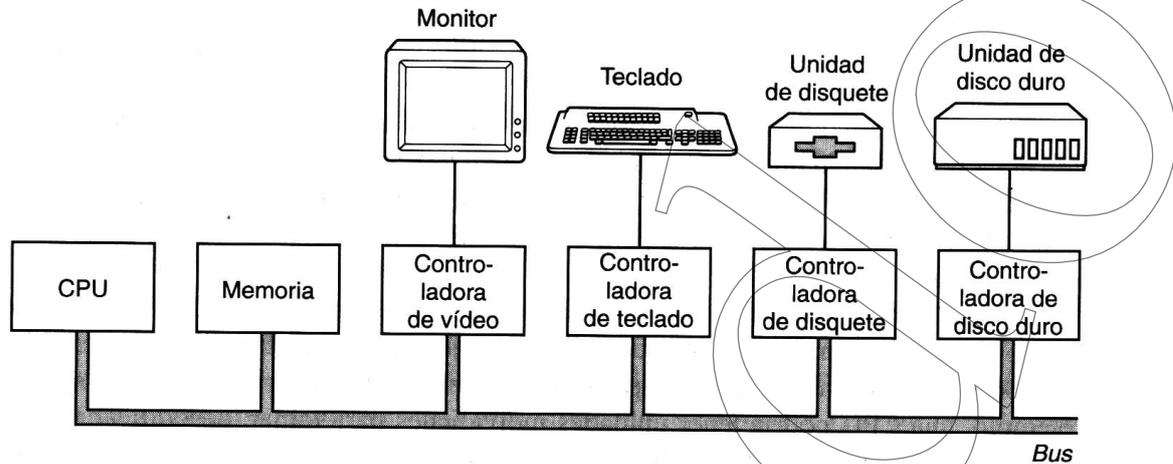


Ilustración 1-1: Arquitectura típica de un ordenador.

El bus es un conjunto de líneas digitales por el que se transfiere, en grupos de 8, 16, 32 o incluso 64 bits la información que se transfiere. Esta información fluye entre CPU y memoria, o entre CPU y los dispositivos. En el caso más que probable de que el sistema disponga de un dispositivo de acceso directo a memoria (DMA) la información también puede fluir directamente de los dispositivos a memoria y viceversa.

En un bus encontramos tres tipos de líneas: líneas de direcciones, líneas de datos, y líneas de control. Cuando por ejemplo, la CPU necesita escribir un dato en memoria, coloca en las líneas de dirección la dirección en la que desea escribir el dato. A continuación, coloca el dato en las líneas de datos, y mediante las líneas de control, indica que desea realizar una escritura, que dicha escritura es en memoria, y cuándo debe comenzar y terminar dicha transferencia.

En la mayoría de los sistemas, por cuestiones de eficiencia, no existe un único bus, sino que hay buses especializados en la conexión de dispositivos. En el capítulo dedicado a la gestión de la lectura/escritura encontrará más detalles sobre esto.

La memoria puede ser de dos tipos: memoria de lectura y escritura (o memoria RAM, Random Access Memory), o memoria de sólo escritura (o memoria ROM, Read Only Memory), en cuyo caso, su contenido, que no se pierde al retirar la alimentación a la máquina, no se puede modificar. Existen distintos tipos de memoria de sólo lectura en función de la tecnología que utilicen (PROM, EPROM, EEPROM) pero a efectos de esta asignatura las trataremos a todas las memorias ROM de la misma manera. En ambos casos, a la información se accede mediante la dirección que ocupa en la memoria. La cantidad de memoria que es capaz de direccionar un sistema se calcula elevando 2 al número de líneas de direcciones existente en el bus (lo cual, normalmente, viene impuesto por el procesador). Muy frecuentemente, en el sistema no se instala toda la memoria que este es capaz de direccionar; en el tema dedicado a la memoria virtual encontrará más detalles sobre esto.

Los dispositivos se conectan al bus (si bien pueden usar un bus específico para ellos) y la CPU puede acceder a ellos o bien mediante instrucciones específicas de acceso a dispositivos (lo que se llama acceso a dispositivos mapeados en bus de entrada/salida, mecanismo habitual por ejemplo de los procesadores Intel) o bien accediendo a ellos como si fueran posiciones de memoria (lo que se llama acceso a dispositivos mapeados en memoria, habitual en los procesadores Motorola).

En los epígrafes siguientes, se profundiza en algunos aspectos de especial relevancia para el estudio de los siguientes temas.

1.1.1.1 La CPU. Modos de ejecución

La CPU contiene un conjunto de registros de un número determinado de bits, normalmente 32, y ejecuta instrucciones muy simples que suelen consistir en mover y hacer operaciones aritméticas o lógicas entre los datos almacenados en sus registros y la memoria. El conjunto de valores de los registros de la CPU constituye el estado de la CPU en un momento dado. De estos registros, hay dos de especial importancia, que son el que contiene la dirección de la próxima instrucción a ejecutar (llamado frecuentemente IP, de Instruction Pointer, o PC, de Program Counter) y el llamado registro de estado (frecuentemente llamado CCR, de Condition Code Register), que contiene una serie de bits que indican entre otras cosas si la última operación aritmética produjo resultado positivo, negativo, cero, además de información relativa al estado actual del procesador.

Para facilitar la construcción de sistemas operativos robustos, la mayoría de los procesadores modernos tienen al menos dos modos de ejecución, a los que llamaremos genéricamente modo supervisor y modo usuario, si bien cada fabricante de procesadores pueda darles nombres distintos.

La CPU arranca siempre en modo supervisor, y mientras se encuentra en este modo, puede ejecutar cualquier instrucción de su conjunto de instrucciones, y acceder a cualquier dirección de memoria o de entrada/salida sin ningún tipo de restricciones. Es por ello que cuando se ejecuta código del sistema operativo la CPU suele estar en este modo.

Por el contrario, cuando la CPU está en modo usuario, hay determinadas instrucciones que no se pueden ejecutar, y posiblemente tampoco se pueda acceder a determinadas direcciones de memoria ni a ningún dispositivo de entrada/salida (se dice que los recursos a los que no se tiene acceso son recursos privilegiados). Para construir un sistema seguro, se debe garantizar que siempre que se ejecuta código ajeno al sistema operativo la CPU se debe encontrar en modo usuario, garantizándose así que es sólo el sistema operativo quien puede acceder a los recursos privilegiados, como por ejemplo, los dispositivos de entrada/salida. De esta forma, se obliga al usuario a acceder a dichos recursos a través del sistema operativo. Si mientras está la CPU en modo usuario intenta ejecutar una instrucción no permitida en este modo, se considera que se ha producido un fallo de protección, lo que provoca una excepción. El concepto de excepción se detalla en el apartado Interrupciones y excepciones 1.1.1.2.

En la práctica, la mayoría de los procesadores tienen varios modos intermedios entre los dos modos expuestos. Algunos procesadores, como los procesadores Pentium, permiten incluso configurar en qué modo se deben ejecutar determinadas operaciones. No obstante, para la construcción de un sistema operativo seguro, es suficiente con los

dos modos expuestos, por lo que no entraremos en particularidades específicas de ningún procesador.

1.1.1.2 Interrupciones y excepciones

Una interrupción es una alteración en el flujo de ejecución de instrucciones del procesador, que puede venir desencadenada por tres causas:

- Una interrupción hardware: todos los procesadores tienen varias líneas hardware de interrupción (entradas de interrupción), que se suelen emplear para facilitar el control de los dispositivos de entrada/salida. Cuando un dispositivo requiere atención por parte del procesador, activa una señal en dicha entrada cambiando su estado de 0 a 1 o viceversa (dependiendo del tipo de CPU, o de cómo esté configurada la entrada de interrupción), lo cual provoca una interrupción.
- Una excepción: se denomina así al resultado de un intento fallido de ejecución de una instrucción. Por ejemplo, el intento de ejecución en modo usuario de una instrucción no permitida en dicho modo, o un intento de acceso a una dirección de memoria no válida.
- Ejecución de una instrucción de petición de interrupción: todos los procesadores tienen en su conjunto de instrucciones una instrucción para forzar por programa una interrupción. Nombres típicos para estas instrucciones son INT, TRAP... normalmente, la instrucción tiene como parámetro el número identificador de la interrupción que se quiere forzar.

Sea cual sea el origen de la interrupción, el tratamiento que se le da es el mismo¹:

1. Normalmente, la CPU pasa a modo supervisor, y se almacena en la pila el estado de la CPU, o al menos los registros más importantes de la misma.
2. Determina la dirección en la que se debe encontrar la rutina de servicio de interrupción (también conocida como SSI, Subrutina de Servicio de Interrupción). Para determinar esta dirección, los procesadores suelen asociar un número a cada interrupción, que en el caso de las interrupciones ocasionadas por programa, coincide con el parámetro de la instrucción de petición de interrupción). La forma más frecuente de usar este dato es utilizándolo para indexar una tabla de rutinas de interrupción, tabla que contiene en su entrada i -ésima la dirección en que se encuentra la SSI de la interrupción i .
3. Una vez determinada la dirección de la SSI, se salta a dicha dirección, es decir, se ejecuta la rutina de interrupción. Dicha rutina termina siempre con una instrucción especial, que se suele llamar RTI (retorno de interrupción) o RTE (retorno de excepción), que tiene como efecto los siguientes pasos:

¹ Si bien hay que aclarar que en el caso de las interrupciones hardware, hay que tener en cuenta que la interrupción puede estar enmascarada, lo cual quiere decir que no se atenderá hasta que no se desenmascare. Estos detalles, no obstante, no nos van a interesar de momento.

4. Se recupera de la pila el estado previo de la CPU, volviendo esta al modo de ejecución (supervisor o usuario) en que se encontrase previamente.

Nótese que, especialmente en el caso de las interrupciones hardware, estas permiten la ejecución de una subrutina para llevar a cabo una actividad en cualquier momento que sea necesario, siendo ello transparente para la actividad que en dicho momento estuviese llevando a cabo la CPU.

1.1.1.3 Arranque del sistema

El arranque del sistema, también conocido como boot, es la secuencia de acciones por las que el sistema, tras ser conectado o reiniciado, llega a estar operativo. El proceso de arranque consta de las siguientes etapas:

1. Cuando un procesador comienza a recibir energía, o cuando se activa su entrada de RESET, comienza una secuencia de inicialización interna. El último paso de dicha secuencia de inicialización consiste en comenzar a ejecutar instrucciones a partir de una dirección de memoria determinada. La forma en que se determina dicha dirección depende de cada procesador.
2. Lo que ocurre a partir de dicho momento, depende de dónde se encuentra el sistema operativo:
 - En los sistemas más simples, el sistema operativo se encuentra en memoria no volátil (ROM, PROM, EPROM, EEPROM, FLASH, etcétera). En dicho caso, el código que se está ejecutando ya pertenece al propio sistema operativo, por lo que comienza directamente la fase de inicialización del sistema operativo.
 - Lo más habitual es que el sistema operativo resida en algún soporte externo a la memoria (disco, servidor de red), y por tanto deba ser cargado en memoria. En este caso, lo que se comienza a ejecutar es un cargador hardware² del que la máquina dispone en memoria no volátil, que realiza las siguientes acciones:
 - Realiza una inicialización mínima de los dispositivos del sistema: determina sus características (cantidad de memoria, dispositivos conectados), comprueba que el sistema funciona, y determina de qué dispositivo se debe cargar el sistema operativo, y
 - lee de dicho dispositivo (frecuentemente, un disco) un programa cargador, perteneciente ya al sistema operativo, que es cargado en memoria y al cual se le transfiere el control del sistema.

Muy frecuentemente, este cargador forma parte de un conjunto de rutinas almacenadas en memoria no volátil llamadas BIOS (Basic

² Llamado así para distinguirlo del cargador del sistema operativo, que se describe a continuación

Input-Output System). Estas rutinas facilitan un manejo mínimo de los dispositivos de la máquina.

3. El cargador del sistema operativo (que en el caso de residir éste en disco suele tener un tamaño fijo y ocupar una zona predeterminada del disco) tiene por misión cargar en memoria el sistema operativo y transferir el control a éste una vez realizada la carga. La forma en que se carga el sistema operativo depende de cada caso.
4. Una vez cargado el sistema operativo, comienza la fase de inicialización del mismo. Esta fase puede constar de los siguientes pasos:
 - Se crean las estructuras de datos propias del sistema operativo: tabla de PCB's, mapas de bits para la gestión de la memoria, etcétera.
 - Comprobación del sistema. Se completan las pruebas realizadas por el cargador hardware (por ejemplo, detección de dispositivos plug'n play) y opcionalmente, si procede, se comprueba que los sistemas de archivos estén en un estado coherente. Esta última operación, dado que suele ser costosa en tiempo, se lleva a cabo sólo en caso de que hayan indicios que apunten a lo contrario.
 - Se carga en memoria cualquier componente del sistema operativo (por ejemplo, gestor de dispositivos) que falte por cargar y se crean los procesos del sistema necesarios.
 - Se crea uno o más procesos de inicio o proceso de login. De estos procesos se crea uno por consola o terminal, y permiten iniciar sesión a los usuarios. Cuando un usuario inicia sesión, el proceso de login lanza un intérprete de comandos para el usuario, intérprete que puede ser tanto gráfico como textual. En algunos sistemas, se crea directamente el intérprete de comandos, si no es necesario verificar la identidad del usuario.

1.1.2 Procesos

Un proceso se puede definir simplemente como un programa en ejecución, si bien veremos en el capítulo dedicado a procesos que esta definición habrá de ser matizada. A la vista de esta definición, podemos definir un sistema operativo multiprogramado como un sistema que permite la ejecución simultánea³ de más de un proceso. Los sistemas operativos multiprogramados hacen uso de mecanismos de protección para dotar a cada proceso de su propio espacio de memoria, aislando así a cada proceso de los demás e impidiendo por tanto que un proceso pueda interferir en la correcta ejecución de otros procesos o incluso del propio sistema operativo.

³ Como se verá en el capítulo dedicado a procesos, el término "ejecución simultánea" puede tener varios significados. Por el momento, admitiremos la idea intuitiva de que hay varios procesos en memoria y en ejecución.

A la vista de la definición que hemos hecho del término proceso, hemos de hacer notar que:

- Si se ejecuta simultáneamente dos o más veces un mismo programa, cada ejecución constituye un proceso distinto. Esto es el caso que se da cuando varios usuarios conectados por ejemplo a una misma máquina UNIX, hacen uso del editor del sistema para modificar sus archivos de texto. El programa es el todos los casos el mismo (emacs, por ejemplo) pero cada usuario ejecuta su propio proceso.
- Un proceso puede mediante una llamada al sistema específica solicitar que se sustituya el código del programa que está ejecutando por otro programa (es el caso de la llamada `execve` en UNIX). Sería por ejemplo el caso de que tras editar un programa, el editor transfiriese el control al compilador para compilarlo. En este caso, a pesar de que el programa que se ejecuta cambia, el proceso sigue siendo el mismo.

Algunas cosas a tener en cuenta también en los sistemas multiprogramados son las siguientes:

- Los sistemas proporcionan siempre mecanismos mediante los cuales un proceso puede crear un nuevo proceso, normalmente, una copia de sí mismo. En dicho caso, el sistema suele considerar una relación paterno-filial entre el proceso creado y su creador (se dice que el proceso creador es el proceso padre del proceso creado), concediéndose determinados privilegios del proceso padre sobre el proceso hijo.
- Dado que los sistemas garantizan aislamiento entre los procesos, también proporcionan siempre mecanismos de comunicación y sincronización entre estos.
- El sistema siempre proporciona algún mecanismo de identificación de procesos. Si bien en algunos sistemas, como es el caso de RTE, identifican los procesos mediante literales alfanuméricos, lo más común es que la identificación de procesos se lleve a cabo mediante valores numéricos enteros. A un identificador de proceso se le suele llamar PID (Process Identifier).

1.1.3 Llamadas al sistema

Una llamada al sistema es una petición que hace un proceso al sistema operativo para obtener cualquier tipo de servicio. La interfaz que el sistema operativo presenta a los programas viene dada por el conjunto de todas las llamadas al sistema de las que dispone. A esta interfaz es a lo que se llama API (Application Programming Interface). A continuación, se describe este concepto, así como los métodos mediante los cuales se pueden implementar las llamadas al sistema.

1.1.3.1 Concepto de API

Como se acaba de describir, el API de un sistema operativo es la interfaz de programación que este presenta a los programas, y viene dada por el conjunto de todas sus llamadas al sistema. Dado que normalmente los procesos de usuario no tendrán acceso directo a los recursos de la máquina, toda la interacción con esta se realizará siempre a través de llamadas al sistema. Por tanto, los programas se hacen independientes de la máquina, aunque dependientes del API del sistema operativo para el que se han construido.

Es por esto último por lo que se han desarrollado algunas APIs estandarizadas, de forma que los programas que escribamos haciendo uso de dichas APIs serán compatibles con todos los sistemas operativos que las implementen. El mejor ejemplo de ello es POSIX (Portable Operating System Interface, la X final va de regalo), un estándar propuesto por ISO que ha sido adoptado en la mayoría de los sistemas UNIX. Otros sistemas operativos, como es el caso de los sistemas de Microsoft, tienen sus propios APIs que gozan de gran popularidad. Este es el caso de WIN32, el API de los sistemas Windows, y gracias al cual, las dos familias de productos Windows (las derivadas de Windows 95 y las derivadas de Windows NT) son compatibles entre sí a pesar de ser sistemas operativos completamente diferentes. No obstante, tanto Windows 2000 como Windows XP implementan también la interfaz POSIX.

1.1.3.2 Métodos de Implementación

Las llamadas al sistema se pueden implementar de dos formas: mediante llamadas a rutinas o mediante interrupciones software, concepto descrito en el epígrafe 1.1.1.2.

La implementación mediante rutinas más simple sería aquella en la que por cada llamada al sistema hay una rutina a la que los procesos pueden llamar (mediante una instrucción CALL o equivalente). No obstante, esto tendría una importante complejidad para los procesos, pues deberían conocer la dirección en la que se encuentra cada una de estas rutinas, pues, hemos de tener en cuenta que estas rutinas son externas a los procesos, pues pertenecen al sistema operativo. Es por ello por lo que normalmente hay un punto de entrada único al sistema operativo, es decir, hay una única rutina a la cual se invoca para solicitar cualquier llamada al sistema; la forma en la que se seleccionan las distintas llamadas al sistema es mediante un parámetro que identifica una llamada en cuestión.

Este método, a pesar de ser simple, tiene serios inconvenientes:

- Es necesario hacer una llamada a una rutina externa al programa. Es por tanto necesario conocer a priori la dirección en que se encuentra dicha rutina, o al menos, el procedimiento por el cual se determina. Las soluciones más frecuentes a este problema son:
 - La rutina se encuentra en una dirección fija conocida de antemano. El problema de esta solución es que hace a los programas dependientes de esta dirección. Futuras versiones del sistema operativo no pueden cambiar esta dirección si desean mantener la compatibilidad.

- La rutina se encuentra en una dirección cualquiera, y en algún lugar conocido de antemano existe una variable que contiene la dirección de la rutina. Esto da libertad al diseñador del sistema para cambiar en futuras versiones la dirección de la rutina, siempre y cuando mantenga la dirección de la variable que contiene la dirección de la rutina.
- Los programas que escriben los usuarios pueden hacer referencia a esta rutina mediante un nombre simbólico reservado (por ejemplo, en ensamblador la llamada se haría mediante una instrucción `CALL SYS`, donde `SYS` representa la dirección del punto de entrada) y sería el montador de enlace del sistema el que, al construir el ejecutable, sustituiría dicho nombre simbólico por la dirección actual del punto de entrada. Esto no obstante implicaría que en caso de cambio de la dirección de dicho punto de entrada, todos los programas deberían ser recompilados.
- Como se indicó en el epígrafe 1.1.1.1, en un sistema operativo robusto los procesos de usuario se ejecutarán con la CPU en modo usuario, y esta pasará a modo supervisor sólo cuando ejecute código del sistema operativo. Por tanto, cuando se efectúa una llamada al sistema, la CPU debe pasar a modo supervisor. Dado que la instrucción `CALL` no efectúa dicho cambio de modo, esto quiere decir que el cambio de modo y la llamada en sí se deben efectuar por separado. Esto es algo que los procesadores no suelen permitir, pues constituiría un serio problema de seguridad, pues podría dar lugar a que un proceso de usuario consiga poner la CPU en modo supervisor manteniendo el control de la misma.

Es por ello por lo que, excepto para sistemas muy simples en los que no hay mecanismos de protección ni necesidad de mantener la compatibilidad, este método no se suele emplear, sino que se usan excepciones para implementar las llamadas al sistema.

Como se indicó en el apartado 1.1.1.2, una interrupción software se trata por la CPU pasando a modo supervisor, y determinando según algún mecanismo (específico de la propia CPU) la dirección en la que se encuentra la rutina de servicio de interrupción que se ha de ejecutar, ejecutando dicha rutina en modo supervisor a continuación, y recuperando el modo de ejecución previo tras la ejecución de dicha rutina. El diseñador del sistema operativo puede construir el punto de entrada al sistema como una rutina de interrupción y colocarla libremente en cualquier lugar de la memoria, siempre que establezca los mecanismos adecuados para que cuando se solicite una llamada al sistema la CPU determine la dirección de la rutina como rutina de servicio. Para determinar qué servicio del sistema es el que se invoca, se suele pasar un identificador de servicio como parámetro en uno de los registros de la CPU.

Nótese la implementación de las llamadas al sistema mediante interrupciones software tiene las siguientes ventajas sobre el uso de rutinas:

- El diseñador del sistema operativo tiene total libertad a la hora de ubicar en memoria la rutina que sirve como punto de entrada al sistema operativo. Dado que el mecanismo por el que dicha dirección se determina a partir de la interrupción software son dependientes del procesador, los

programas de usuario no dependen de ellos, con lo que es fácil asegurar la compatibilidad entre distintas versiones del sistema operativo.

- La conmutación a modo supervisor por parte de la CPU se hace de forma implícita. Dado que un proceso de usuario, con la CPU en modo usuario, normalmente no podrá alterar las estructuras del sistema en la que se almacenan las direcciones de las rutinas de interrupción, un proceso de usuario no puede poner la CPU en modo supervisor manteniendo el control de ésta.
- Se posibilitan las llamadas implícitas al sistema. Supongamos que un proceso de usuario intentase ejecutar una instrucción no permitida en este modo, por ejemplo, una instrucción INP para acceder a un dispositivo de entrada/salida. Como se indicó en el epígrafe 1.1.1.2, esto provocaría una excepción que se trataría de forma análoga a una interrupción software. La rutina de interrupción que tratase esta excepción podría determinar qué pretendía hacer el proceso que ha causado esta excepción y si dicho proceso tendría derecho a hacerlo; en caso de que así fuese, podría simular la ejecución de la instrucción prohibida y retornar el control al proceso, con lo que este tendría la percepción de haber ejecutado la instrucción prohibida. Nótese que esto ha funcionado de forma equivalente a que el proceso hubiese solicitado explícitamente al sistema la ejecución de la actividad.

Es importante hacer notar que en la práctica, cuando escribamos programas que hagan directamente llamadas al sistema, en la mayoría de los casos las llamadas se harán invocando a rutinas a rutinas en C u otro lenguaje de alto nivel en la que el nombre de la rutina coincidirá con el nombre de la llamada (`read`, `fork`, `open`, etcétera), siendo los argumentos de la rutina los parámetros de la llamada. Esto no implica ni mucho que las llamadas en dicho sistema se implementen mediante rutinas; simplemente quiere decir que el fabricante del compilador que usamos se ha tomado la molestia de escribir para cada llamada al sistema una rutina que admite como argumentos los parámetros de la llamada, y se encarga de efectuar la llamada al sistema escribiendo su código en el registro oportuno de la CPU, pasando los demás parámetros a través de la pila u otros registros, y provocando la interrupción.

1.1.4 Usuarios

Muy frecuentemente, los sistemas multiprogramados mantienen una asociación entre cada proceso y el usuario al cual pertenece dicho proceso, recibiendo los permisos para acceder a los distintos recursos del sistema en función de dicho propietario.

La forma en la que se suele identificar a los usuarios es mediante identificadores numéricos (llamados normalmente UID, User Identifier). Esto puede contrastar con el hecho de que, por ejemplo en UNIX, hacemos referencia a los usuarios mediante un identificador alfanumérico. La explicación es que dicho identificador alfabético es simplemente un dato más que el sistema asocia a cada usuario, si bien la identificación interna es mediante un UID numérico.

Normalmente, se permite también definir grupos a los que se pueden adscribir los usuarios. Los grupos se suelen identificar mediante un identificador numérico de

grupo (llamado normalmente GID, Group Identifier), si bien suelen tener siempre asociado un nombre de grupo para facilitar su comprensión por parte de los usuarios.

De esta forma, los permisos que reciben los procesos dependerán de la pareja [UID, GID], es decir, de quién es su usuario y a qué grupo pertenece. Normalmente, tanto cada UID como cada GID son únicos en el sistema.

1.1.5 Archivos

La práctica totalidad de los sistemas permiten organizar la información en archivos para su tratamiento. Por ello, los sistemas operativos cuentan con numerosas llamadas en su API para el tratamiento de archivos: creación, apertura, lectura, búsqueda directa, escritura, cierre, etcétera.

Los archivos residen normalmente en dispositivos de almacenamiento como discos magnéticos, discos ópticos, unidades de cintas, o unidades de memoria externa. Para facilitar la organización del sistema de archivos, se suele establecer una organización jerárquica basada en directorios y subdirectorios, para lo cual las API de los sistemas proporcionan también llamadas para manipulación del sistema de archivos: creación de directorios, subdirectorios, copia y movimiento de archivos, cambio de nombre de archivos y directorios, etcétera.

La forma en que se identifica el dispositivo en el que reside un archivo varía mucho dependiendo de cada sistema operativo. Por ejemplo, en los sistemas de Microsoft los dispositivos reciben nombres del tipo A:, B:, C:, etcétera, para las unidades de disco (otros nombres de dispositivos son PRN para la impresora por defecto, LPT1: para el dispositivo conectado al puerto paralelo 1, COM1: para el dispositivo conectado al puerto serie 1...). Otros sistemas, como UNIX, asocian a cada dispositivo un archivo especial dentro de un directorio del sistema de archivos (directorio /dev) y ocultan los dispositivos sobre los que residen los archivos mediante un árbol de directorio único, permitiendo montar el sistema de archivos que contiene un dispositivo dado sobre un directorio vacío del sistema de archivos.

En la Ilustración 1-2 se muestra a la izquierda el árbol de directorio de un sistema, en el que en la raíz existen dos directorios a y b, conteniendo el directorio a dos subdirectorios c y d, y estando el directorio b vacío. En el centro, un dispositivo contiene un sistema de archivos que en su raíz contiene dos subdirectorios x e y. A continuación, en la derecha, el sistema de archivos del dispositivo se ha montado sobre el directorio b, dando como resultando el árbol único de directorios que se muestra.

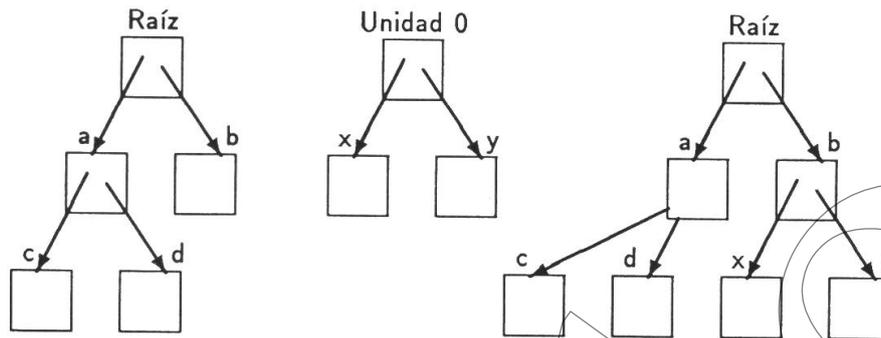


Ilustración 1-2: Montaje de archivos en un sistema UNIX

En los sistemas multiusuario, normalmente el sistema operativo proporciona mecanismos de protección del sistema de archivos, basándose en establecer permisos de acceso a los archivos en función de la identidad del propietario de cada archivo y del grupo al que pertenece, y de la identidad y grupo al que pertenece el propietario del proceso que desea acceder al archivo.

1.1.6 Intérpretes de órdenes

Un intérprete de órdenes, también conocido como intérprete de comandos o shell, es simplemente un programa interactivo que lee comandos del usuario, y los ejecuta efectuando si es necesario para ello las llamadas al sistema oportunas. Este es normalmente el primer proceso que se lanza cuando un usuario inicia sesión.

Algunos sistemas operativos, como es el caso de UNIX, permiten que el usuario establezca el intérprete de órdenes que prefiera de entre un amplio abanico de intérpretes disponibles (ksh, bash, tcsh, csh, por citar los más populares) o incluso puede usar un intérprete construido por él mismo. En otros sistemas operativos, el intérprete de órdenes viene integrado como parte del sistema y el usuario no tiene posibilidad de cambiarlo.

Algunas funciones que la mayoría de los intérpretes de comando soportan son:

- Ejecución de un programa, redireccionando su entrada o su salida desde o hacia un archivo:

```
sort <arch-entrada >arch-salida
```

- Ejecución en paralelo de varios procesos unidos por una tubería (redirección de la salida del primero hacia la entrada del segundo):

```
who | wc -l
```

- Ejecución de archivos conteniendo secuencias de comandos, que en muchos casos llegan a constituir verdaderos programas (también conocidos como scripts).

1.1.7 Interfaces gráficas

Una interfaz gráfica es simplemente un intérprete de órdenes en modo gráfico, en lugar de textual. Es decir, el usuario, en lugar de teclear un comando de la forma:

```
copy \documentos\f1.doc \usuarios
```

simplemente señalará con el puntero del ratón a un icono representando el archivo `f1.doc` y manteniendo pulsada la tecla control y el botón izquierdo del ratón, lo arrastrará colocándolo sobre un icono que representa al directorio `\usuarios`, dejando allí el archivo simplemente soltando la tecla control y el botón del ratón. Este tipo de intérpretes goza de una gran popularidad hoy en día por su facilidad de uso, ya que para un usuario no experto su manejo resulta mucho más intuitivo que tener que aprender un glosario de comandos y la sintaxis por la que estos se construyen (si bien, frecuentemente este método suele ser más rápido una vez que se sabe manejar).

Muy frecuentemente, las interfaces gráficas ofrecen un API que permite la construcción de programas haciendo uso de sus recursos. De esta manera, las aplicaciones que se ejecutan desde la misma tienen un aspecto coherente con la interfaz gráfica del propio sistema (lo que en inglés se denomina el *look and feel* del sistema). Este es por ejemplo el caso de los sistemas Windows de Microsoft, en el que buena parte del API Win32 está dedicada al manejo de la interfaz gráfica. Es por ello por lo que todos los programas que se ejecutan en Windows usan los mismos tipos de ventanas y con la misma personalización.

1.2 Modelos de diseño

A continuación, vamos a mostrar las distintas estrategias de diseño que se pueden seguir a la hora de construir un sistema operativo. Para ello, vamos a describir los modelos de organización interna más habituales en la construcción de sistemas operativos. Las tres primeras organizaciones que vamos a explicar (epígrafes 1.2.1, 1.2.2 y 1.2.3) son modelos de diseño específicos para construir sistemas operativos, mientras que el modelo de máquinas virtuales (epígrafe 1.2.4) es un concepto fuertemente relacionado con el de sistema operativo, aunque un monitor de máquinas virtuales no es un sistema operativo propiamente dicho; por su parte, los sistemas orientados a objetos (epígrafe 1.2.5) consisten simplemente en la incorporación del análisis y el diseño orientado a objetos a la construcción de un sistema operativo, como cualquier otro proyecto software.

No obstante, los sistemas operativos que usamos en la práctica no tienen que corresponderse al cien por cien con solo uno de estos modelos. Frecuentemente, los sistemas operativos son híbridos: se puede considerar que un sistema se rige fundamentalmente por uno de los modelos, aunque puede incorporar ideas de los demás.

1.2.1 Modelo monolítico

Un sistema operativo con organización monolítica se caracteriza porque todo el sistema operativo se encuentra en un único espacio de memoria (podríamos decir que constituye un único ejecutable). Esto tiene como consecuencia que:

- Cualquier rutina del sistema operativo puede invocar a cualquier otra rutina, simplemente haciendo una llamada directa (mediante un CALL), y
- Cualquier rutina del sistema operativo puede manipular directamente cualquier estructura de datos de cualquier módulo del sistema.

Esto conlleva dos importantes ventajas:

- Eficiencia en tiempo, pues si un componente del sistema operativo necesita servicio de cualquier otro componente, puede hacer una invocación directa mediante llamada a subrutina, o si necesita cualquier información puede acceder a ella directamente, y
- Eficiencia en ocupación de memoria, pues al ser accesibles todas las rutinas y todas las estructuras de datos desde cualquier módulo del sistema operativo, no es necesario replicar código ni datos, cosa que sí puede ocurrir en otros modelos (véase 1.2.3).

Es por ello por lo que este modelo de diseño tiende a producir sistemas compactos y rápidos. No obstante, estos sistemas presentan también los siguientes inconvenientes:

- Complejidad y dificultad de mantenimiento, pues normalmente existen muchas interdependencias entre los distintos módulos del sistema, y
- Dificultad de depuración, ya que un error en un módulo del sistema, puede ocasionar un fallo en cualquier otro módulo (efectos colaterales). Este sería el caso, por ejemplo, en que debido a un error en el gestor de disco, un puntero que apunta a donde menos debería hacerlo modifica accidentalmente datos que pertenecen al administrador de memoria, y es este último quien manifiesta el fallo.

1.2.1.1 Estructura de un sistema operativo monolítico

En la Ilustración 1-3 se muestra la estructura de un sistema operativo con organización monolítica. Esta consta de:

- Módulo despachador (conocido también por el término inglés dispatcher): Es el punto de entrada al sistema operativo (véase epígrafe 1.1.3.2). Dicho punto de entrada es una rutina a la que transfiere el control cada vez que:
 - Un proceso hace una llamada al sistema
 - Un proceso de usuario ejecuta una instrucción prohibida, o se produce cualquier otro fallo de protección.

Su función es determinar la causa que ha provocado su ejecución y seleccionar la rutina de servicio que debe atenderla. Su funcionamiento se detalla un poco más adelante.

- Rutinas de servicio: son las rutinas que implementan cada uno de los servicios del sistema operativo.
- Rutinas auxiliares: son rutinas que se encargan de tareas auxiliares y que pueden ser de utilidad para cualquier otra rutina (de servicio o auxiliar) del sistema operativo: manipulación de estructuras de datos, comprobaciones habituales, etcétera.

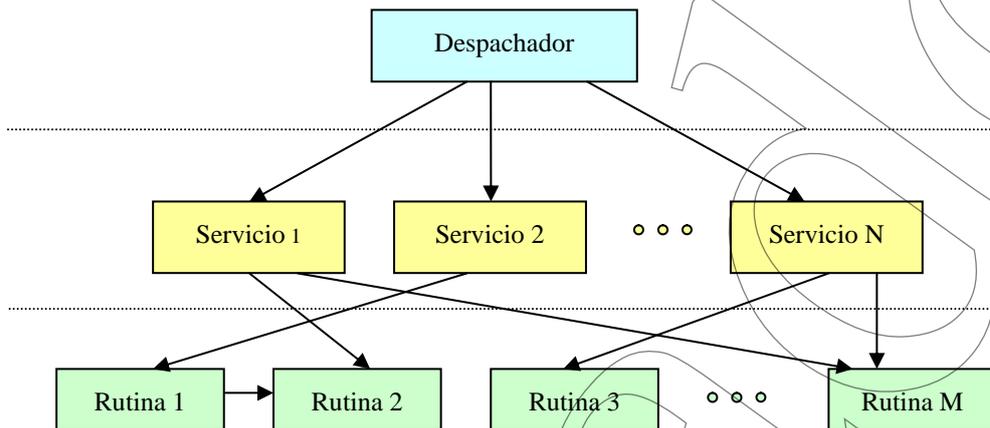


Ilustración 1-3: Estructura de un sistema operativo monolítico

Es importante hacer notar que, en paralelo con esto, se pueden ejecutar una serie de rutinas de interrupción invocadas directamente por el hardware, dedicadas normalmente a la gestión de dispositivos.

Respecto al funcionamiento del despachador, ya se ha indicado que su función es determinar cómo se ha de tratar la causa que ha motivado su ejecución. En primer lugar, determinará si el control le ha sido transferido por un fallo de protección o por una llamada al sistema realizada por un proceso. En el primer caso, determinará de qué fallo se trata y qué proceso lo ha ocasionado, y decidirá la acción a llevar a cabo, que podría ser simular una llamada al sistema (véase epígrafe 1.1.3.2 en lo relativo a las llamadas implícitas) o tratar lo acontecido como un error, abortando posiblemente el proceso causante. En caso de que la ejecución del despachador se deba a una llamada al sistema por parte de un proceso, las acciones a llevar a cabo son:

1. Obtener el identificador del servicio solicitado y sus parámetros.
2. Comprobar que estos son válidos.
3. Determinar la rutina que atiende el servicio solicitado, e invocarla.
4. Devolver el control

Es importante recordar que, debido a que el despachador se ejecuta en respuesta o bien a una interrupción software o bien a una excepción, el procesador conmutará automáticamente a modo supervisor al comenzar a ejecutar su código, y volverá al modo en que se encontrase previamente al devolverse el control.

Para determinar la dirección de la rutina de servicio que se ha de invocar, en lugar de usar una estructura de comprobaciones anidadas como la siguiente:

```
if(ident_servicio == IDENT1)
    servicio1(...)
else if (ident_servicio == IDENT2)
    servicio2(...)
else if (ident_servicio == IDENT3)
    servicio3(...)
...
```

que obliga a efectuar sucesivas comprobaciones, un método más eficiente que se usa consiste en utilizar el identificador de servicio para indexar una tabla en la que la entrada *i*—ésima contiene la dirección de la rutina de servicio cuyo identificador de servicio es *i*. De esta forma, seleccionar e invocar a la rutina de servicio es algo tan simple y eficiente como:

```
rutina= tabla_rutinas_servicio[ident_servicio];
rutina(...);
```

Ejemplo de sistemas operativos que emplean esta organización interna son UNIX y la mayoría de sus clones, como puede ser el caso de LINUX. SOLARIS es una excepción, pues tiene organización micronúcleo.

1.2.2 Modelo en estratos

Este modelo parte de la idea de que colocando una capa de software sobre el hardware desnudo mostramos a las capas superiores de software una máquina ampliada con una interfaz de más alto nivel. De esta forma, el sistema operativo se puede construir apilando capas de software de manera que cada capa resuelve un problema concreto, ofreciendo una interfaz a la capa inmediatamente superior en la que dicho problema queda resuelto.

Por ejemplo, como se muestra en la Ilustración 1-4, sobre el hardware desnudo podríamos colocar una primera capa que implemente la multiprogramación. De esta manera, lo que se coloque sobre esta capa verá una máquina multiprogramada. Sobre esta primera capa, podemos colocar una segunda capa que gestione la memoria y el disco. De esta forma, lo que se coloque sobre esta capa tendrá la percepción de ejecutarse sobre una máquina multiprogramada en la que cada proceso tiene su propio espacio de memoria. Sobre esta capa, se puede colocar otra que gestione la comunicación con la consola, con lo que las capas superiores verán una máquina multiprogramada, con protección de memoria y en la que cada proceso tendrá su propia consola virtual. Posteriormente, se puede añadir una nueva capa que gestione la entrada/salida sobre los dispositivos, lo que añade a las anteriores capas la posibilidad de que cada proceso tenga sus propios dispositivos abstractos de entrada/salida. Sobre esta capa, podrían ejecutarse ya los procesos de usuarios.

Un sistema operativo que empleó una organización similar a la descrita fue el sistema operativo THE, diseñado por Dijkstra en 1968. Otro sistema operativo histórico que usó una organización por estratos fue MULTICS (véase epígrafe ¡Error! No se encuentra el origen de la referencia.). Este sistema se consideraba organizado en anillos (capas, al fin y al cabo) de forma que cada anillo se correspondía con un modo de ejecución del procesador (véase epígrafe 1.1.1.1), siendo el anillo más interior el que se

corresponde con el mayor nivel de privilegio, y el anillo exterior el que se corresponde con un menor nivel de privilegio. De esta manera, los propios mecanismos de protección del hardware garantizaban que desde cada anillo se podía acceder directamente a rutinas y recursos de los anillos más exteriores, pues requerían menor nivel de privilegio, pero nunca a los recursos situados en anillos más interiores, ya que requerían un mayor nivel de privilegio. En caso de que desde un anillo se intentase acceder a un recurso en un anillo más interior, esto provocaba una excepción que era capturada por el núcleo del sistema (anillo más interno). El intento se analizaba y, en caso de considerarse permitido, se transfería el control al anillo al que se pretendía acceder, realizándose de esta forma una llamada implícita como se describe en el epígrafe 1.1.3.2). Un sistema reciente que utiliza organización por capas es OS/2.

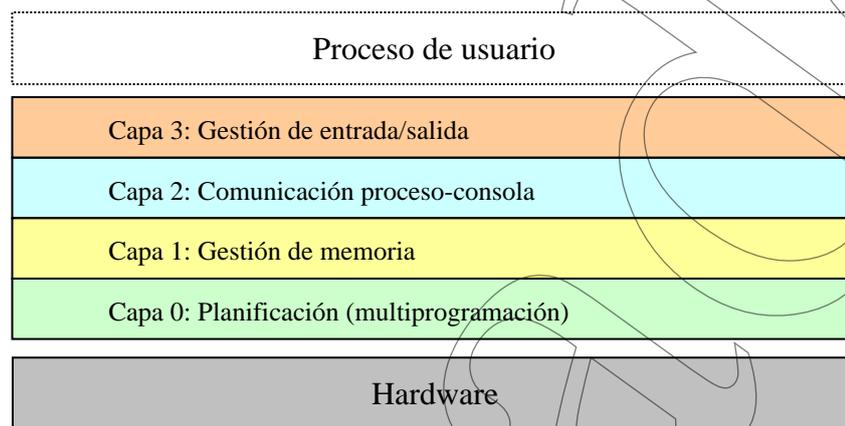


Ilustración 1-4: Ejemplo de sistema organizado por estratos

1.2.3 Modelo micronúcleo

Este modelo responde a la tendencia de los sistemas operativos modernos en hacer el núcleo del sistema operativo tan pequeño como sea posible, ejecutando parte del propio sistema operativo como procesos de usuario.

Un sistema que siga este modelo de forma estricta, constaría de un núcleo mínimo que se encargaría de solamente de:

- Gestión de la multiprogramación.
- Comunicación entre procesos.
- Atención de interrupciones.

De esta forma, proporciona la infraestructura mínima necesaria para que el resto de los componentes del sistema (gestores de dispositivo, administración de archivos, administración de memoria, etcétera) se implementen como procesos separados. Como estos procesos se ejecutan con total aislamiento entre sí (véase epígrafe 1.1.2) la única forma de interacción entre estos es mediante los mecanismos de comunicación entre procesos establecidos por el núcleo del sistema.

El funcionamiento de un sistema operativo de este tipo se ilustra en la Ilustración 1-5. Si por ejemplo un proceso de usuario necesita leer información de un archivo, envía a través de los mecanismos de comunicación entre procesos del núcleo una petición

dirigida al administrador de archivos, que como se puede ver es un proceso más en el sistema, y queda a la espera de respuesta. Normalmente, estas peticiones se efectúan en forma de mensajes. Para atender dicha petición, el administrador de archivos necesitará el servicio del gestor de disco, que será el encargado de enfrentarse a los detalles particulares del disco en el que reside la información. Para ello, envía de la misma forma una (o varias) petición/es al gestor de discos, el cual es otro proceso en el sistema. Cuando el gestor de disco ha realizado cada actividad solicitada por el administrador de archivo, lo comunica enviándole un mensaje de respuesta. A su misma vez, cuando el administrador de archivo ha satisfecho la petición realizada por el proceso de usuario, envía al proceso de usuario la respuesta que este esperaba. Es por este mecanismo de petición/respuesta por lo que en algunos libros se refieren a este modelo como modelo cliente/servidor.

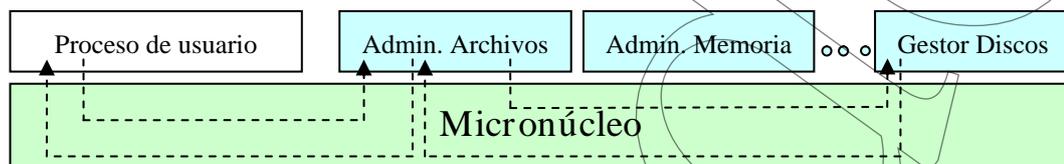


Ilustración 1-5: Sistema operativo con organización micronúcleo

Las principales características de este modelo son la modularidad y el aislamiento entre sus módulos componentes. Como consecuencia de estas características, estos sistemas tienen las siguientes ventajas:

- Facilidad de depuración y puesta a punto, pues el núcleo es más simple y el resto de los componentes se ejecutan como procesos aislados lo que facilita su depurado de forma aislada al resto del sistema.
- Robustez, entendiéndose por robustez que fallos en un componente del sistema no deben causar efectos colaterales en otros componentes, ya que cada componente es un proceso que no tiene acceso al espacio de memoria (ni a código ni a datos) de los demás componentes del sistema.
- Flexibilidad: es fácil sustituir un módulo por otro, incluso se puede hacer en caliente (sin reiniciar el sistema) simplemente deteniendo y terminando un proceso y lanzando otro.
- Fácil adaptación a sistemas distribuidos, ya que la comunicación entre módulos se hace mediante mensajes, y esta primitiva de comunicación es fácilmente adaptable a un sistema distribuido.

No obstante, como consecuencia del aislamiento entre sus módulos, tiene las siguientes desventajas:

- La interacción entre los componentes del sistema mediante mecanismos de comunicación entre procesos (normalmente, mensajes) es mucho más lenta que la interacción mediante llamada directa a rutina, como se hace por ejemplo en los sistemas monolíticos. El coste de una llamada directa a rutina son unas cuantas instrucciones para colocar los parámetros en la pila más la propia instrucción CALL, mientras que enviar un mensaje a otro proceso conlleva una llamada al sistema (implica al núcleo), lo que

puede conllevar decenas e incluso centenares de instrucciones. Respecto a los parámetros de la llamada, es necesario llevar a cabo un trasvase de información ya que por ejemplo, el proceso cliente no puede enviar al proceso servidor un puntero a los parámetros en su espacio de memoria. Debido a esto, estos sistemas tienden a ser más lentos.

- Dado que cada componente del sistema se ejecuta como proceso aislado, no es posible compartir rutinas de uso común, así como tampoco es posible que varios componentes puedan acceder a estructuras de datos compartidas. Es por ello por lo que frecuentemente las rutinas comunes están replicadas en cada componente, y por lo que muchas estructuras de datos están también replicados entre componentes, teniendo cada uno una copia de la información, y siendo necesario además mantener la coherencia entre todas las copias. Esto hace que estos sistemas, además de ser más lentos, tiendan a ocupar más memoria.

1.2.4 Sistemas de máquinas virtuales

Un sistema de máquinas virtuales no es un sistema operativo propiamente dicho, sino una herramienta útil para ejecutar uno o más sistema operativo; a su vez, nos proporciona algunos principios que sí se pueden aplicar en el desarrollo de determinadas partes de un sistema operativo.

Nuevamente, recurrimos a la idea de que colocando una capa de software sobre el hardware desnudo ofrecemos a las capas superiores una máquina ampliada con una interfaz de más alto nivel. Tan sólo que ahora replanteamos: ¿Y por qué de más alto nivel? ¿Y por qué una sola máquina?

En efecto, la idea que subyace es que podemos aplicar una capa relativamente simple de software sobre el hardware desnudo que simule varias máquinas (máquinas virtuales) idénticas a la máquina real. A dicha capa software es a lo que se llama un monitor de máquinas virtuales. De esta forma, todo programa que se pueda ejecutar sobre la máquina real se puede ejecutar también sobre cualquier de las máquinas virtuales. En particular, podríamos ejecutar sobre una máquina virtual cualquier sistema operativo que se pueda ejecutar sobre la máquina real. De esta forma, podemos ejecutar varios sistemas operativos simultáneamente sobre la máquina real, como se muestra en la Ilustración 1-6, que muestra la arquitectura de un sistema de máquinas virtuales.

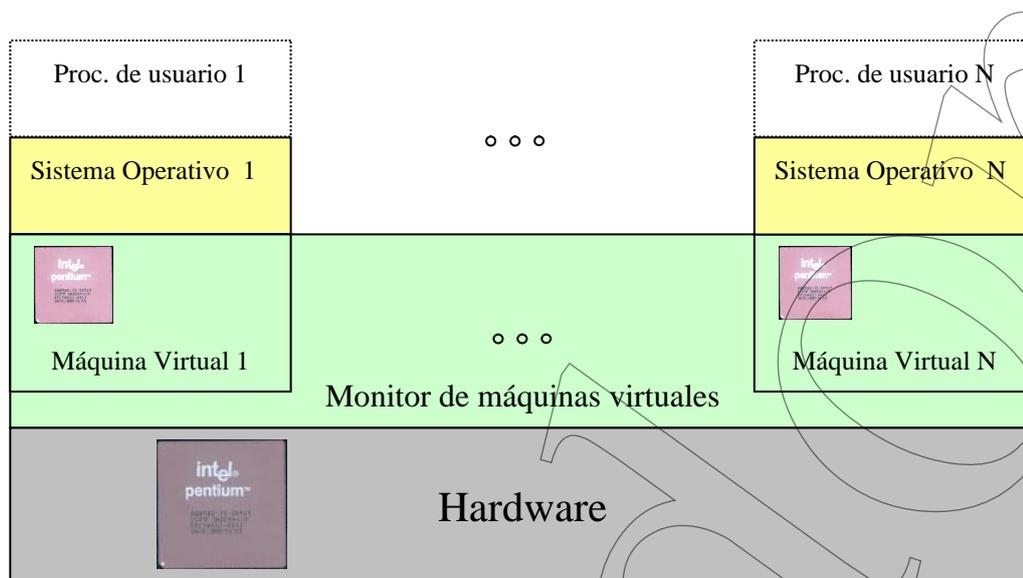


Ilustración 1-6: Esquema de un sistema de máquinas virtuales.

En un sistema de máquinas virtuales, la CPU sólo se encuentra en modo supervisor cuando ejecuta código del monitor de máquinas virtuales. Ahora bien, dado que el monitor simula máquinas virtuales idénticas a la real, simula también que cada una de estas máquinas tuviese su propia CPU, y simula también los modos de ejecución de éstas. Es decir, cada máquina virtual, tiene su CPU virtual que puede estar en modo virtual supervisor o en modo virtual usuario. De esta forma, mientras se ejecuta código de un sistema operativo sobre una máquina virtual, la CPU real está en modo usuario, pero la CPU virtual de la máquina en cuestión está en modo virtual supervisor. Cuando se ejecuta código de un proceso de usuario que se ejecuta sobre un sistema operativo de una máquina virtual, ambos procesadores, virtual y real, se encuentran en modo usuario.

En esta situación, supongamos que un proceso de usuario que se ejecuta sobre un sistema operativo de una máquina virtual intentase ejecutar una instrucción privilegiada. En esta situación, la CPU real está en modo usuario (pues no estaba ejecutando código del monitor) y la CPU virtual está en modo virtual usuario (pues no estaba ejecutando código del sistema operativo). Al estar la CPU real en modo usuario, el intento de ejecución de la instrucción privilegiada provoca un fallo de protección que transfiere el control al monitor de máquinas virtuales. Éste, determina en qué máquina virtual se ha producido el fallo, y su tratamiento dependerá del estado en que se encontrase la CPU virtual de dicha máquina. Dado que la CPU virtual se encontraba en modo virtual usuario, simula un fallo de protección en la máquina virtual en que se produjo, pasando la CPU a modo virtual supervisor y transfiriendo el control a la rutina que el sistema operativo de dicha máquina estableció como rutina de servicio para dicha excepción, volviendo la CPU real a modo usuario. En esta situación, comienza a ejecutarse el código del sistema operativo de la máquina virtual. Muy posiblemente, dicho sistema operativo, que ve su CPU en modo supervisor (virtual), intentará ejecutar instrucciones privilegiadas. Al intentar ejecutarse una de estas instrucciones, se producirán nuevos fallos de protección (pues la CPU real sigue en modo usuario). Estos fallos de protección transferirán nuevamente el control al monitor de máquinas virtuales, el cual, al comprobar que la CPU de la máquina en la que se han producido los fallos está en modo virtual supervisor, simulará la ejecución de las instrucciones privilegiadas con lo que el sistema operativo que se ejecuta en la máquina virtual tiene la percepción de que en efecto, su CPU virtual está en modo supervisor.

Como ventajas, este tipo de sistemas nos permite:

- La ejecución simultánea de varios sistemas operativos en una misma máquina.
- Adaptar cada máquina virtual a las necesidades del usuario.
- Depurar fácilmente un sistema operativo en desarrollo.

Como inconveniente, sólo hay que tener en cuenta que, realmente, el hardware que tenemos es el que tenemos. Si sobre una máquina simulamos un número determinado de máquinas, la potencia (tanto de cálculo como de recursos disponibles) total de todas las máquinas será como mucho igual a la de la máquina real. Normalmente, sensiblemente inferior, debido a la sobrecarga tanto en tiempo como en recursos que introduce el propio monitor.

Un sistema desarrollado tal y como hemos descrito es VM/370. Este fue un sistema presentado por IBM en 1972, y se ejecutaba sobre un IBM System/370, proporcionando la ilusión de disponer de varios System/370 idénticos al original. De hecho, VM/370 era capaz de ejecutarse sobre uno de los System/370 virtuales. Posteriormente, VM/370 fue reemplazado por VM/390. Actualmente, el entorno de virtualización que comercializa IBM es z/VM (<http://www.vm.ibm.com>).

Otros sistemas de máquinas virtuales que podemos encontrar son VMWARE (<http://www.vmware.com>) y PLEX86 (<http://plex86.sourceforge.net>). VMWARE es un producto comercial que simula una máquina virtual dentro de un sistema operativo que puede ser tanto LINUX como Windows 2000/2003/XP. Sobre la máquina virtual simulada se puede ejecutar cualquier sistema operativo que pueda ejecutarse sobre la máquina real. Por su parte, PLEX86 es un entorno de virtualización ligero (muy eficiente) que está pensado exclusivamente para la ejecución de LINUX. Su eficiencia reside precisamente en que se realizan importantes simplificaciones orientando la emulación a la ejecución de un sistema operativo concreto.

1.2.5 Sistemas orientados a objetos

Los sistemas orientados a objetos no constituyen en sí mismos un modelo de diseño de sistemas operativos, sino que son la consecuencia de la aplicación de los métodos de análisis, diseño (y no necesariamente programación) orientados a objetos a la construcción del sistema operativo. Un sistema operativo orientado a objetos puede desarrollarse conforme a cualquiera de los anteriores modelos de diseño, pero puede manejar cada uno de sus recursos internos como objetos, permitiendo sobre cada objeto un conjunto de posibles operaciones en función de su clase.

Un ejemplo de sistema operativo que hace uso de esta filosofía son las distintas versiones de Windows. En Windows, cada recurso del sistema que pueda ser manipulado desde un proceso de usuario (elementos de la interfaz gráfica de usuario, archivos, procesos, etcétera) es un objeto que se identifica en el sistema por lo que se llama un handle (un identificador numérico entero, que actúa como referencia al objeto). Desde los procesos de usuario no se puede acceder a los detalles internos del objeto, sino que el sistema provee una serie de llamadas que, en función de la clase del objeto, permiten manipular sus atributos y efectuar operaciones sobre el mismo. Cada llamada al sistema

que efectúa una operación sobre un objeto lleva como primer argumento el handle del objeto.

1.3 Ejemplos de organización

A continuación se exponen las organizaciones internas de los sistemas operativos que se utilizarán como referencia a lo largo del temario. Para ello, hemos elegido Linux y Windows 2000 por ser los sistemas operativos de más amplia difusión, y MINIX por tratarse de un sistema cuyo propósito es precisamente servir como ejemplo en la construcción de sistemas operativos.

1.3.1 Organización de MINIX

MINIX se estructura como un conjunto de procesos que se comunican entre sí con los procesos de usuario usando paso de mensajes como única primitiva de comunicación. Este diseño se adapta al cien por cien a la organización micrónúcleo descrita en el epígrafe 1.2.3.

Como se muestra en la Ilustración 1-7, MINIX se estructura en cuatro niveles, cada uno de los cuales realiza una función perfectamente definida:

- El nivel inferior (Micrónúcleo) se encarga de la gestión de la multiprogramación, así como del tratamiento mínimo de interrupciones y excepciones, proporcionando a los niveles superiores un modelo de procesos independientes que se comunican por medio de mensajes.
- En el siguiente nivel se encuentran los gestores de dispositivos, también llamados tareas. Estos son procesos (uno por cada tipo de dispositivo) que realizan la gestión a bajo nivel de cada dispositivo. Contienen por tanto todo el código dependiente de dispositivo. Existe a este nivel un proceso especial, llamado Tarea del Sistema, que no se corresponde con ningún dispositivo. Contiene simplemente el código de una serie de servicios del sistema operativo que, en cualquier otro sistema, formaría parte del núcleo. En MINIX, no obstante, y para preservar su organización micrónúcleo pura (por razones lectivas), se sacrifican prestaciones y dicho código se extrae del núcleo y constituye un proceso separado.
- En el tercer nivel se encuentran solamente dos procesos, que son los administradores de memoria y de archivos. El administrador de memoria implementa todas las llamadas al sistema que impliquen gestión de memoria, como `fork`, `exec`, `brk`, etcétera. Por su parte, el administrador de archivos contiene todo el código de entrada/salida independiente de dispositivos, realizando todas las llamadas relativas al sistema de archivos, como `read`, `mount`, `chdir`, etcétera.
- Finalmente, en el último nivel, se encuentran los procesos de usuario. Entre ellos se encuentra el proceso `init`, que es el primer proceso que se ejecuta una vez ha arrancado la máquina, y que tiene como función crear los demás procesos del sistema.

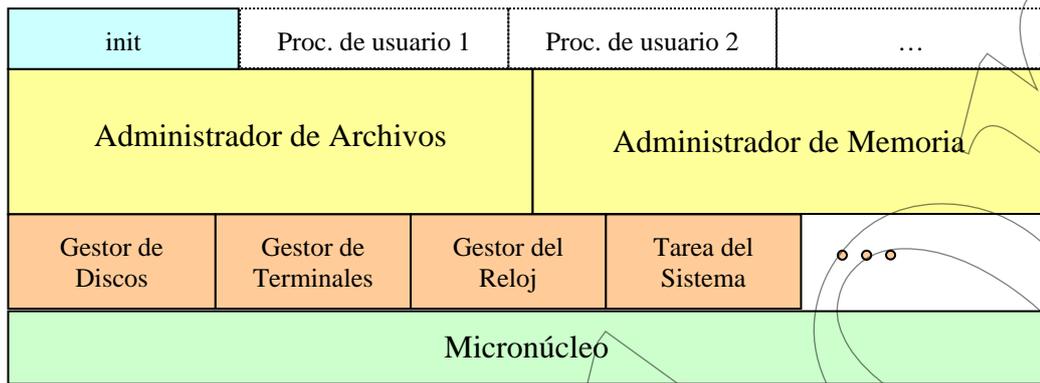


Ilustración 1-7: Organización interna de MINIX

Dado que MINIX se puede ejecutar (en su versión 1.0) incluso sobre un procesador 8086 sin modos de ejecución, en caso de que el procesador disponga de ellos, sólo el micronúcleo y los procesos del segundo nivel (gestores de dispositivos) se ejecutarán en modo supervisor. El resto del sistema se ejecuta en modo usuario.

1.3.2 Organización de Windows 2000

Windows 2000 combina ideas de varios de los modelos de diseño expuestos en el apartado 1.2. Desde el punto de vista de que buena parte de la funcionalidad del núcleo se ha extraído de éste y se implementa como procesos externos al núcleo, se puede considerar un sistema operativo con organización micronúcleo. Ahora bien, a diferencia del modelo micronúcleo puro en el que el núcleo gestiona solamente la multiprogramación, la gestión de interrupciones y la comunicación entre procesos, el núcleo de Windows 2000 incorpora en su propio espacio de memoria la gestión de dispositivos (lo que es propio de una organización monolítica). Esta decisión se ha tomado por cuestiones de eficiencia, ya que a efectos prácticos los gestores de dispositivos tienen un alto grado de dependencia con el núcleo y viceversa; permitir que esta interacción se efectúe mediante acceso directo es mucho más eficiente que efectuarlo mediante primitivas del sistema. Por supuesto, esto implica que un gestor de dispositivo (un driver, en terminología Windows) incorrectamente codificado puede corromper el núcleo y causar la caída del sistema. Los diseñadores de Windows se defienden de esta crítica argumentando que en un sistema con arquitectura micronúcleo pura en la que un gestor de dispositivo fallase, el sistema también terminaría cayendo al no funcionar correctamente el dispositivo gestionado. Además de ello, y dado que desde los procesos de usuario los recursos del sistema se gestionan como objetos, se puede considerar también un sistema orientado a objetos.

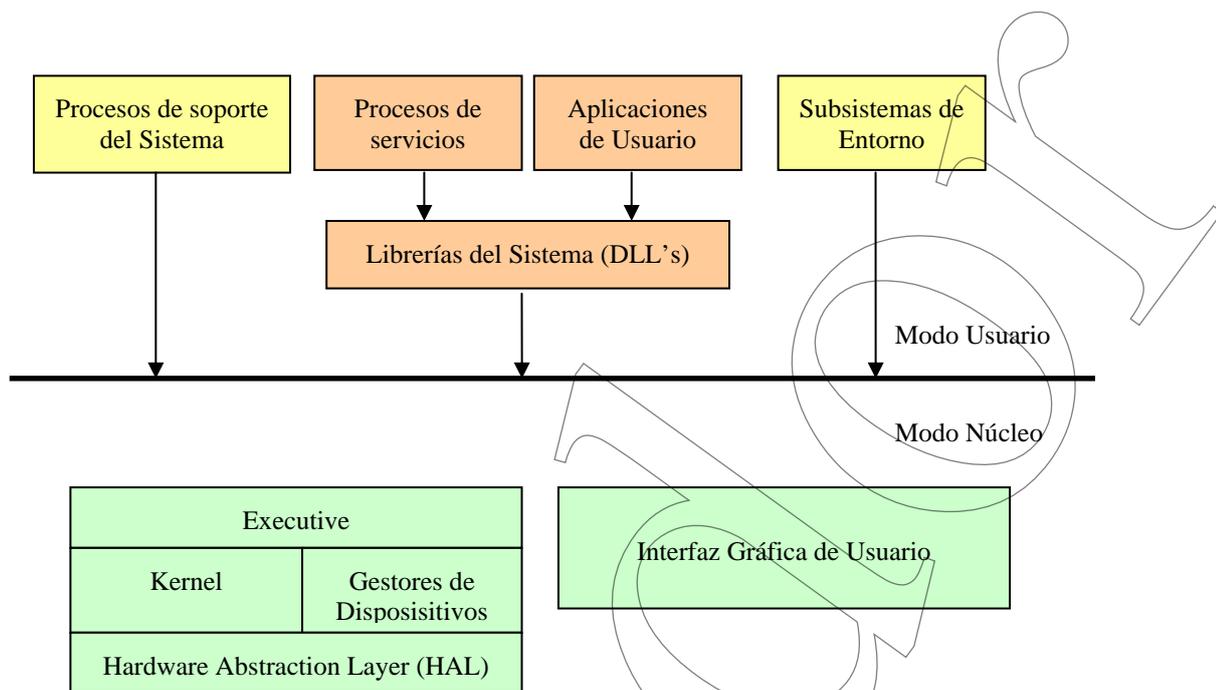


Ilustración 1-8: Organización (simplificada) de Windows 2000

En la Ilustración 1-1 se muestra una simplificación de la organización interna de Windows 2000. En primer lugar, obsérvese la línea que separa la parte que se ejecuta en modo usuario de la parte que se ejecuta en modo núcleo (modo supervisor). Los elementos sobre esta línea se llaman procesos en modo usuario, de los cuales existen los siguientes grupos:

- **Procesos de Servicios:** son procesos que gestionan servicios de la interfaz de programación Win32, tales como la cola de impresión, el cliente DHCP, el cliente DNS, etcétera. Muchas aplicaciones, como SQL Server, Microsoft Exchange Server, por citar sólo un par de ellas, también incluyen componentes que se ejecutan como servicios. La ejecución de estos procesos (cuándo comienzan a ejecutarse y cuándo se detienen) la gestiona un proceso especial llamado service control manager.
- **Procesos de soporte del sistema,** como el proceso de login o el gestor de sesiones. Se diferencian de los procesos de servicios en que no son iniciados por el service control manager.
- **Aplicaciones de usuario,** que pueden ser de cinco tipos, en función del API que empleen: aplicaciones Win32, Windows 3.1, MS-DOS, POSIX o OS/2 1.2.
- **Subsistemas de entorno,** que hacen de interfaz entre las aplicaciones de usuario y el sistema, ocultando la interfaz nativa de éste tras una serie de rutinas que implementan varias interfaces de programación. Concretamente, Windows 2000 implementa de fábrica las interfaces Win32 (común a todos los sistemas Windows), POSIX y OS/2.

También en la parte del sistema que se ejecuta en modo de usuario, obsérvese la caja rotulada como Librerías del Sistema (DLL's). En Windows 2000, las aplicaciones

de usuario no llaman directamente a los servicios nativos del API de Windows, sino que en su lugar lo hacen a través de una o más librerías de enlace dinámico (DLL, de Dynamic Link Library) que proporciona el sistema. La funcionalidad de estas librerías es traducir cada llamada del API que emplea el proceso de usuario (llamadas documentadas por Microsoft) en las llamadas del API interno (indocumentadas) de Windows 2000. Esta traducción puede implicar el envío de un mensaje al subsistema de entorno empleado por la aplicación.

Por su parte, los componentes en modo núcleo de Windows 2000 son:

- **Executive:** lo constituyen los servicios básicos del sistema operativo, lo que incluye gestión de memoria, gestión de hilos y procesos, seguridad, entrada/salida y comunicación entre procesos.
- **Kernel:** esta formado por las funciones de más bajo nivel, tales como planificación de hilos, atención de excepciones e interrupciones, y sincronización entre procesadores (pues Windows 2000 soporta multiprocesamiento simétrico). También proporciona un conjunto de rutinas y objetos básicos que emplea el Executive para implementar construcciones de más alto nivel.
- **Los Gestores de Dispositivos,** que comprenden tanto a los gestores de dispositivos hardware de entrada/salida (traducen peticiones abstractas de entrada/salida a comandos específicos del dispositivo), como a los administradores de los distintos sistemas de archivos soportados y los gestores de red.
- **El Hardware Abstraction Layer (HAL)** es una capa de código que aísla al Kernel, a los gestores de dispositivos y al resto de Windows 2000 de detalles específicos del hardware.
- **La Interfaz Gráfica de Usuario** implementa las funciones del entorno gráfico tales como ventanas, controles, cursores, mapas de bits, y demás recursos de la interfaz de usuario.

Evidentemente, la estructura de Windows 2000 tiene muchos detalles que se omiten en esta visión general. En el texto [Solomon02] puede encontrar una descripción de las interioridades del sistema con gran nivel de detalle.

1.3.3 Organización de Linux

El sistema Linux, como la mayoría de las implementaciones de UNIX, se compone de tres cuerpos principales de código:

- **El núcleo,** que implementa todas las abstracciones del sistema operativo, como es la gestión y planificación de hilos y procesos, la comunicación entre procesos, la administración de archivos, etcétera.
- **Las bibliotecas del sistema,** que definen un conjunto estándar de funciones a través de las cuales las aplicaciones pueden interactuar con el

núcleo y que implementan gran parte de la funcionalidad del sistema operativo que no necesita todos los privilegios del núcleo.

- Las utilidades del sistema, que son programas que realizan tareas de uso y administración; `ls`, `cp`, `su`, o `lpd` son ejemplos de este tipo de herramientas.

Centrándonos en el núcleo, Linux, al igual de nuevo que la mayoría de los sistemas UNIX, es un sistema monolítico, es decir, incluye prácticamente toda la funcionalidad del núcleo en un gran bloque de código que se ejecuta en un único espacio de memoria, por lo que su estructura poco difiere de la que se muestra en la Ilustración 1-3 de la página 15. Por tanto, todos los componentes del núcleo tienen acceso a todas las estructuras de datos y rutinas del mismo. Recordemos del epígrafe 1.2.1 que esto tiene como consecuencia que resulta difícil cualquier modificación del núcleo, como añadir un nuevo controlador de dispositivo o una función del sistema de archivos. Este problema es especialmente crítico para Linux, ya que su desarrollo se lleva a cabo por un grupo libremente asociado de programadores independientes.

Para solucionar este problema, Linux está organizado como un conjunto de bloques independientes denominados módulos cargables. Los módulos cargables de Linux tienen dos características importantes:

- Enlace dinámico: un módulo del núcleo puede cargarse y enlazarse dentro del núcleo, mientras el núcleo permanece en la memoria y en ejecución. Un módulo puede ser también desenlazado y borrado de la memoria en cualquier momento.
- Módulos apilables: los módulos se organizan en una jerarquía. Los módulos individuales actúan como bibliotecas cuando son referenciados por módulos clientes de mayor nivel en la jerarquía y como clientes cuando los referencian módulos de menor nivel.

El enlace dinámico facilita la tarea de configuración y protege la memoria del núcleo. En Linux, un programa de usuario o un usuario puede cargar y descargar explícitamente módulos del núcleo usando las órdenes `insmod` y `rmmmod` o `modprobe`. El núcleo por sí mismo controla la necesidad de funciones particulares y puede cargar y descargar módulos cuando se necesiten. Con módulos apilables, es posible definir las dependencias entre módulos. Esto tiene dos ventajas:

1. El código común de un conjunto de módulos similares (por ejemplo, controladores para un hardware similar) puede ser trasladado a un único módulo, reduciendo la duplicación.
2. El núcleo puede asegurar que los módulos necesarios estén presentes, absteniéndose de descargar un módulo del que dependen otros módulos en ejecución, y cargando cualquier módulo adicional necesario cuando se cargue un módulo nuevo.

[Loye05] es un texto dedicado íntegramente a describir los detalles e interioridades del núcleo de Linux. Si en su lugar prefiere un resumen algo más extenso que lo que

hemos expuesto aquí de las características más relevantes del núcleo de Linux, puede consultar en [Silberstchaz99] el capítulo dedicado a Linux.

1.4 Para ampliar conocimientos

Si bien existen textos especializados en arquitectura de ordenadores, [Carretero02] dedica su primer capítulo a los conceptos fundamentales sobre ordenadores. Le puede resultar interesante consultar este capítulo en caso de no haber estudiado aún arquitectura de ordenadores en ninguna otra asignatura de su titulación.

El libro de prácticas del mismo autor, [Carretero02b], propone diversas prácticas consistentes en la construcción de partes de un sistema operativo, basándose en un entorno de virtualización que se entrega como material de apoyo. Se recomienda la lectura de los apartados en que se explica el funcionamiento del entorno de virtualización, que le será útil para conocer detalles que no se llegan a exponer en el epígrafe 1.2.4. También se recomienda la realización de la primera práctica, que consiste en la creación de una nueva llamada al sistema, para adquirir experiencia en el manejo de los aspectos expuestos en el epígrafe 1.1.3.2.