Tema 1

Introducción a los Sistemas Operativos

Un elemento esencial en todo sistema de computación es el sistema operativo. En este capítulo veremos cuál es su función y objetivo principal: abstraer el hardware para ofrecer un conjunto de servicios a los usuarios. Con el fin de comprender mejor los distintos elementos que componen un sistema operativo contemporáneo y sus objetivos, veremos como a lo largo de la historia se han ido introduciendo innovaciones para aumentar el rendimiento del sistema.

1.1. ¿Qué es un sistema operativo?

Todo sistema de computación consta de una parte software y otra hardware. El software del computador puede dividirse de modo general en dos clases:

- los **programas del sistema**, que controlan el funcionamiento del computador.
- los **programas de aplicación**, que resuelven los problemas de los usuarios y son específicos para un tipo de problema.

Respecto al hardware, un computador es un sistema complejo que consta de uno o más procesadores, memoria central, relojes, terminales, discos y otros dispositivos de E/S (ver apéndice **??**). Si los programadores tuvieran que tener en cuenta cómo funciona cada uno de estos elementos a la hora de hacer un programa, la elaboración de uno de ellos sería una tarea bastante difícil.

Para evitar que los programadores tengan que enfrentarse a la complejidad del hardware, se introduce una capa software por encima del primero, con objeto de manejar todas las partes del sistema y presentar al usuario una interfaz que sea más fácil de entender y programar. Esta capa es el **sistema operativo** (figura 1.1).

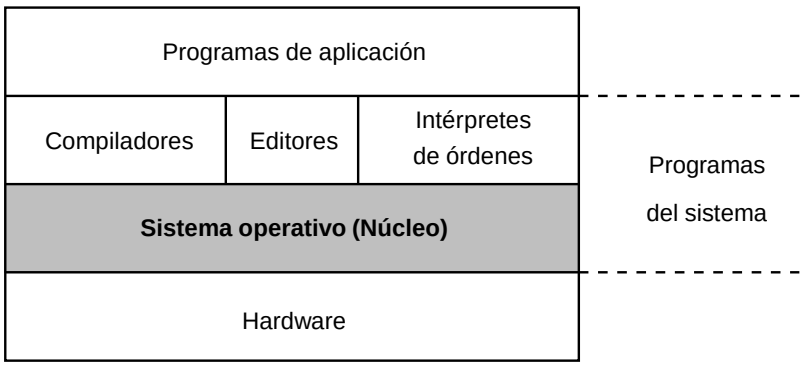

Figura 1.1: Estructura de un sistema de computación

El sistema operativo es la parte más importante de lo que hemos llamado programas del sistema, ya que controla todos los recursos del computador y ofrece la base sobre la cual pueden escribirse los programas de aplicación.

Sobre la capa software que constituye el sistema operativo descansa el resto del software del sistema. Aquí se encuentran el intérprete de órdenes, los compiladores, editores de texto, etc. Estos programas se suelen distribuir junto con el sistema operativo para otorgarle una mayor funcionalidad y comodidad al entorno, por lo que los usuarios pueden creer que forman

parte de él. Esto no es así, la diferencia fundamental que existe entre ambos es que el sistema operativo se ejecuta en **modo núcleo**, mientras que los compiladores y editores funcionan en **modo usuario**. El modo núcleo es un modo de ejecución privilegiado que le permite tener acceso a todo el hardware, a la vez que lo protege de la manipulación indebida por parte del usuario. Si a uno no le agrada un compilador determinado tiene la libertad de escribir uno propio si así lo desea, pero no la tiene para escribir su propio manejador de interrupciones del disco, porque es parte del sistema operativo y está protegido por el hardware contra intentos de modificación por parte del usuario.

Se denomina **núcleo** del sistema operativo a la parte que se comunica directamente con el hardware, éste será el verdadero corazón y motor del sistema.

La pregunta ¿qué es un sistema operativo? es difícil de contestar; debido principalmente a que éstos realizan dos funciones que no están relacionadas entre sí. Para intentar responderla veamos qué es lo que hace un sistema operativo.

Los programas que se ejecutan en el sistema necesitan utilizar recursos. Desde el punto de vista del programa lo ideal sería disponer de todos los recursos del sistema, esto implicaría que no pudiese coexistir junto con otros. Como se verá más adelante en este capítulo, para mejorar el rendimiento del sistema es conveniente que haya varios programas activos simultáneamente. Para conseguir esto es necesario que se repartan los recursos entre todos los programas que compiten por ellos. Esta función la realiza el sistema operativo. Así, podemos decir que éste se comporta como **administrador de los recursos** del sistema, distribuyéndolos entre los programas que se ejecutan en él. Es decir, el sistema operativo concede o deniega las peticiones de recursos que hacen los programas en ejecución.

La gestión de los recursos que realiza el sistema operativo consigue que los programas en ejecución tengan la ilusión de que están solos en el sistema, disponiendo de todos los recursos de éste. En realidad, lo que existe es un reparto transparente de los recursos entre todos los programas que compiten por ellos.

Como podemos ver en la figura 1.1, el sistema operativo se sitúa entre el hardware y el resto del software del sistema, para recibir las peticiones de los distintos programas y controlar de esta forma la asignación de los recursos. Esto añade una funcionalidad más al sistema operativo, que es la de proporcionar una capa de **abstracción del hardware**. La arquitectura de

muchos computadores en el nivel del lenguaje máquina es primitiva y difícil de programar, especialmente lo referido a las operaciones de E/S, debido a la gran variedad de dispositivos existentes. Sería muy complicado para un programador introducirse en los detalles reales de cómo se realiza una operación de lectura o escritura en un disco, por ejemplo. Además, el simple hecho de cambiar de dispositivo implicaría modificar el código del programa. El sistema operativo se encarga de abstraer estos detalles. En el caso de los discos, el sistema operativo presenta la abstracción de que éstos contienen información almacenada como un conjunto de ficheros con un nombre. Para poder acceder a la información sólo se necesita saber el nombre del fichero, y no las operaciones reales que deben realizarse con el disco.

De esta forma, la función del sistema operativo es la de presentar al usuario una máquina ampliada que sea más fácil de programar que el hardware implícito. Es decir, actúa como una interfaz para las distintas peticiones del software.

1.2. Evolución histórica de los sistemas operativos

Para llegar a entender mejor qué son y qué hacen los sistemas operativos modernos, es conveniente considerar la evolución que han sufrido a lo largo de su historia. El estudio de esta evolución nos va a conducir a través de los problemas que se planteaban y las soluciones presentadas, lo que nos permitirá entender mejor cómo y por qué se han desarrollado de la forma que lo han hecho.

Los sistemas operativos y la arquitectura de los computadores se han influido mucho mutuamente. Por un lado, los sistemas operativos se han desarrollado para facilitar el uso del hardware y, por otro, a medida que se iban diseñando nuevos sistemas operativos y se planteaban nuevos problemas, se hacía evidente que ciertos cambios en el diseño del hardware podían solucionar los problemas existentes o bien hacer más simple el sistema operativo.

1.2.1. Los primeros sistemas

Los primeros computadores eran máquinas enormes, ocupaban cuartos enteros y contenían decenas de miles de tubos de vacío, pero eran mucho

más lentos que el más barato de los computadores domésticos que existe hoy.

En estos primeros días un grupo de personas diseñaba, construía, programaba y mantenía cada máquina. La programación se realizaba en lenguaje máquina, ya que no existía todavía ningún lenguaje de programación de alto nivel. Cuando se quería ejecutar un programa había que cargarlo manualmente en memoria; esto se hacía al principio desde un panel con clavijas y posteriormente mediante tarjetas o cintas perforadas. Después había que pulsar los botones adecuados para cargar la dirección de comienzo y empezar la ejecución del programa. A medida que el programa se ejecutaba, el programador podía seguir su ejecución mediante las luces que se encendían en la consola. Si se descubría algún error, el programador podía parar la ejecución del programa, examinar el contenido de la memoria y de los registros del procesador, y depurar el programa directamente desde la consola. La salida del programa se imprimía o se obtenía en tarjetas o cintas perforadas. En este tipo de sistemas la memoria estaba ocupada únicamente por el programa que se estaba ejecutando; eran sistemas **monoprogramados**.

Un aspecto a destacar de esta forma de trabajo era su naturaleza **interactiva**. El programador era también el operador del sistema. Cada programador se reservaba el uso exclusivo de la máquina durante un tiempo determinado; se trataba de un **sistema monousuario**. Este tipo de acceso al computador puede provocar ciertos problemas. Supongamos que reservamos una hora de uso del computador para ejecutar un programa que hemos desarrollado. Si nos encontramos con un error difícil de detectar y no somos capaces de finalizar la ejecución durante dicho tiempo, como probablemente alguien habrá reservado el uso de la máquina a continuación, tendremos que parar, recoger toda la información que podamos y volver más tarde para continuar. Por otra parte, si las cosas nos van bien y terminamos antes de lo previsto, durante el resto del tiempo reservado la máquina permanecerá ociosa.

Con el tiempo se desarrollaron hardware y software adicional. Aparecieron dispositivos como las lectoras de tarjetas y las unidades de cinta magnética. Se desarrollaron los ensambladores, cargadores y enlazadores, para facilitar las tareas de programación, así como las bibliotecas de funciones comunes, que pueden ser utilizadas por cualquier programa sin tener que escribirlas de nuevo.

Las rutinas que realizan las operaciones de E/S son especialmente importantes. Cada dispositivo de E/S tiene sus propias características, por lo que hay que escribir una rutina específica para él. Estas rutinas, llamadas **manejadores de dispositivos** (*device drivers*), se colocan en bibliotecas de

uso común para no tener que ser escritas para cada programa.

Más tarde aparecieron los compiladores de Fortran, Cobol y otros lenguajes. Esto facilitó mucho la tarea del programador, sin embargo hizo más compleja la tarea del computador. Por ejemplo, para preparar un programa escrito en Fortran para su ejecución, el programador tiene que cargar primero el compilador de Fortran en el computador (normalmente el compilador estaba almacenado en cinta magnética). El programa puede ser leído por una lectora de tarjetas y pasado a cinta magnética. El compilador de Fortran produce una salida en lenguaje ensamblador que necesita ser ensamblada con el ensamblador, lo que requiere montar otra cinta con el ensamblador. La salida del ensamblador necesita ser enlazada con las rutinas del sistema adecuadas. Finalmente, la forma binaria del programa está lista para ser ejecutada. Puede ser cargada en memoria y depurada desde la consola como antes. A este conjunto de labores que hay que realizar para ejecutar un programa se le conoce con el nombre de **trabajo**.

1.2.2. Sistemas por lotes

El tiempo adicional que se gastaba en la preparación de los trabajos constituía un problema, ya que durante el tiempo en que se están montando las cintas o el programador está operando en la consola, la CPU permanece ociosa. En estos primeros tiempos había pocos computadores y eran muy caros, por lo que su tiempo era muy valioso y sus propietarios querían que fuesen usados al máximo.

La solución dada a este problema fue doble. Por un lado, aparece el operador profesional y el programador ya no interviene en la operación de la máquina. Así, tan pronto como un trabajo finaliza, el operador arranca el siguiente; por tanto, no hay ya tiempo ocioso debido a que se reserve un cierto tiempo de uso del computador. Al tener el operador más experiencia en el montaje de cintas que el programador, el tiempo de preparación también se reduce. El usuario le proporcionará todas las tarjetas perforadas o cintas que se van a necesitar, así como una corta descripción de cómo debe ser ejecutado el trabajo. Por supuesto, el operador no se encarga de depurar un programa incorrecto, puesto que él no tiene por qué entender el programa. Cuando un programa falla, obtiene un volcado de la memoria que entrega al programador. Así, el operador podrá continuar con el siguiente trabajo. Sin embargo, al programador se le hace más difícil depurar su programa.

También se consigue un ahorro en el tiempo que se dedica a la prepa-

ración de los trabajos reuniendo aquellos que tengan necesidades similares en un lote y procesándolos como un grupo. Esto es lo que se conoce como **procesamiento por lotes** (*batch*). Por ejemplo, supongamos que el operador recibe un trabajo en Fortran, un trabajo en Cobol y otro en Fortran. Si los ejecuta en ese orden tendrá que preparar el computador para ejecutar el trabajo en Fortran, luego para el trabajo en Cobol y después otra vez para el otro trabajo en Fortran. Sin embargo, si ejecuta los dos trabajos en Fortran como un lote sólo tendría que preparar el computador una vez para los dos, con lo cual se ahorra tiempo.

Los cambios introducidos, haciendo que la persona que opera con la máquina sea distinta del usuario y el procesamiento por lotes de trabajos similares, modifican la forma de comunicación entre el programador y la máquina perdiendo su naturaleza interactiva. Estos cambios mejoraron un poco la utilización de los computadores, pero todavía seguía habiendo problemas. Por ejemplo, cuando un trabajo se para, el operador tiene que darse cuenta observando la consola, determinar por qué se ha parado (si es normal o no), hacer un volcado de la memoria si es necesario, y preparar el lector de tarjetas o de cintas con el trabajo siguiente, volviendo a arrancar el computador. Durante esta transición de un trabajo a otro la CPU permanece sin hacer nada.

Para suprimir este tiempo ocioso, se introdujo un secuenciador automático de trabajos, y con él, se creó el primer sistema operativo. Lo que se quería era un procedimiento para transferir automáticamente el control de un trabajo al próximo. Con este propósito se crea un pequeño programa llamado **monitor residente**, que siempre está (residente) en memoria. Como puede verse en la figura 1.2, ahora la memoria contiene al monitor residente y al programa que el usuario manda a ejecutar.

El monitor residente tiene el control del computador cuando éste se enciende, y este control puede transferirse a un programa cuando sea necesario. Cuando termine su ejecución devuelve el control al monitor residente. Por tanto, éste se encarga de transferir el control de un programa a otro dentro del mismo trabajo, y de un trabajo a otro.

¿Pero cómo sabe el monitor residente qué programa ejecutar? Previamente, el operador le dará al monitor residente una breve descripción de qué programas se van a ejecutar y con qué datos, para ello introduce unas **tarjetas de control**. La idea es simple, además de las tarjetas que contienen el programa y las de los datos, se introducen unas tarjetas especiales que contienen instrucciones para el monitor residente indicándole qué programa va a ejecutar. Para distinguir estas tarjetas de control de las de datos o programa,

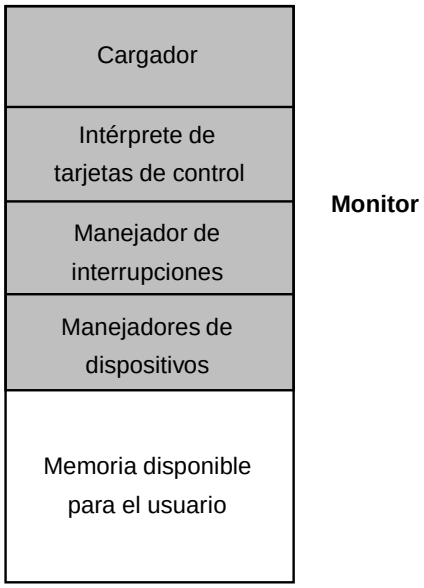

Figura 1.2: Estructura de la memoria con monitor residente

se las identifica con un carácter especial, tal como $, //, etc. En la figura 1.3 se muestra el conjunto de tarjetas necesarias para ejecutar un trabajo.

En un monitor residente se pueden distinguir varios componentes (figura 1.2). Uno es el intérprete de tarjetas de control que es el encargado de traducir cada orden en una ejecutable en código máquina. Éste llamará cada cierto tiempo al cargador para cargar los programas en memoria. Tanto el intérprete de tarjetas de control como el cargador necesitarán realizar operaciones de E/S, por tanto el monitor residente tiene un conjunto de manejadores de dispositivos para trabajar con los dispositivos de E/S del sistema. También se necesita un módulo para el tratamiento de las interrupciones producidas por los dispositivos de E/S, el **manejador de interrupciones**.

El monitor residente proporcionaba una secuenciación automática de los trabajos mediante las indicaciones de las tarjetas de control. Cuando una tarjeta de control indica que se debe ejecutar un programa, el monitor lo carga en memoria y le transfiere el control. Cuando el programa termina, devuelve el control al monitor, que lee la próxima tarjeta de control, carga el programa apropiado y sigue. Esto se repite hasta que todas las tarjetas de control de ese trabajo han sido interpretadas. Entonces el monitor continúa automáticamente con el próximo trabajo.

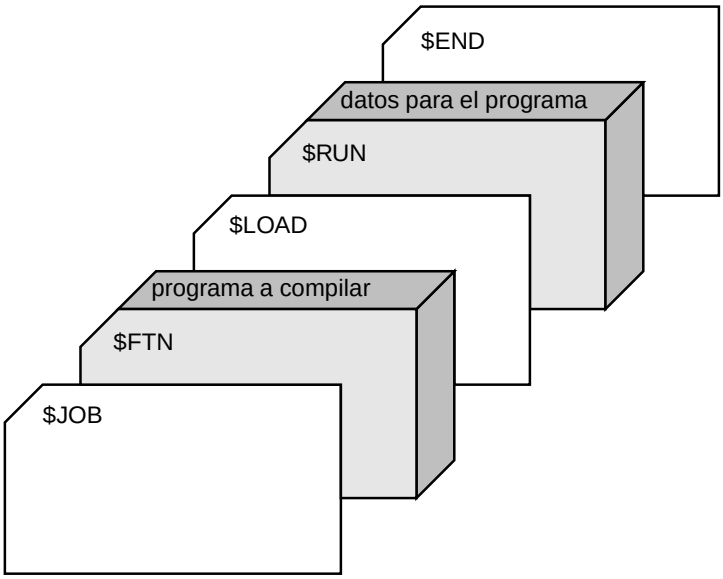

Figura 1.3: Conjunto de tarjetas de un trabajo

Un **sistema operativo por lotes** normalmente lee un flujo de trabajos (de una lectora de tarjetas, por ejemplo), cada uno con sus propias tarjetas de control que predefinen lo que hace el trabajo. Cuando el trabajo termina, normalmente su salida se imprime. La característica principal de un sistema por lotes es la falta de interacción entre el usuario y el trabajo mientras éste se está ejecutando. El trabajo se prepara y se envía; algún tiempo después (minutos, horas o días), aparece la salida. La demora entre el momento de envío del trabajo y su terminación, que se conoce como **tiempo de retorno**, es debida a la cantidad de computación que éste necesita y al tiempo que se tarda en empezar a procesarlo.

La introducción del monitor residente hace necesaria la introducción de diversos mecanismos hardware de protección, para evitar que un programa de usuario acceda a la zona de memoria del monitor, o que monopolice el uso del procesador, por ejemplo. En el apartado 1.7.1 se tratarán los mecanismos necesarios.

El cambio a sistemas por lotes con secuenciación automática de trabajos se hizo para mejorar el rendimiento de la CPU. Pero aún así hay momentos en que ésta permanece ociosa. El problema es debido a que la velocidad de los dispositivos mecánicos de E/S es intrínsecamente más lenta que la de los dispositivos electrónicos. Una CPU lenta puede ejecutar millones de instruc-

ciones por segundo, mientras que una lectora de tarjetas rápida puede leer unas 1200 tarjetas por minuto. Con el tiempo, los progresos de la tecnología producen dispositivos de E/S más rápidos. Pero las velocidades de la CPU también aumentan, incluso de forma más rápida; por tanto, el problema no sólo no queda resuelto sino que incluso empeora.

1.2.2.1. Procesamiento fuera de línea

Una solución a este problema fue reemplazar las lectoras de tarjetas (dispositivos de entrada) y las impresoras de línea (dispositivos de salida) por unidades de cinta magnética. Ahora, en vez de leer directamente las tarjetas, éstas se copian primero en cinta magnética. Cuando un programa necesita tomar su entrada de una tarjeta, el registro equivalente se lee de la cinta. De forma similar, la salida de los programas se escribe en cinta magnética y el contenido de ésta se imprime más tarde. Las lectoras de tarjetas y las impresoras operaban **fuera de línea**, es decir, sin utilizar el computador principal (figura 1.4).

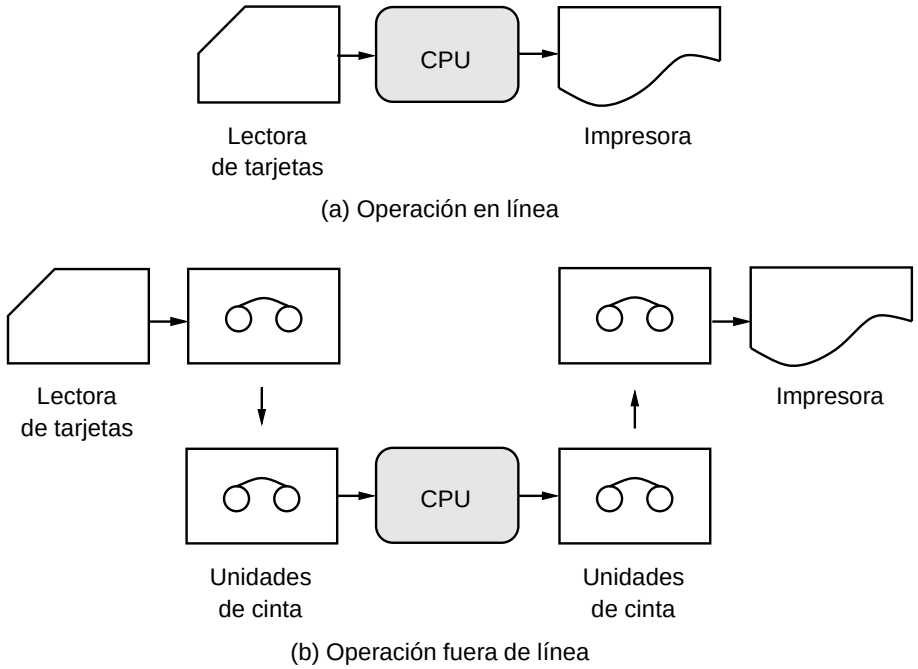

Figura 1.4: Modos de operación

La ventaja principal de la operación fuera de línea es que permite utilizar

múltiples sistemas lectora-a-cinta y cinta-a-impresora para una CPU. De esta forma, si la CPU puede procesar la entrada dos veces más rápido que una lectora de tarjetas, entonces dos lectoras trabajando simultáneamente pueden producir bastantes cintas para mantener a la CPU ocupada. Por tanto, el computador principal ya no está coartado por la velocidad de las lectoras de tarjetas e impresoras de línea, sino por la velocidad de las unidades de cinta magnética, que son mucho más rápidas. Sin embargo, ahora hay una demora aún mayor a la hora de ejecutar un trabajo particular. Primero debe ser pasado a cinta y hay que esperar que haya bastantes trabajos hasta llenarla. Una vez llena tiene que ser rebobinada, desmontada, llevada a la CPU y montada en una unidad de cinta libre.

1.2.2.2. *Buffering*

El procesamiento fuera de línea permite un solapamiento de las operaciones de la CPU y de E/S mediante la ejecución de estas acciones en dos máquinas independientes. Si nos fijamos en la ejecución de un trabajo observaremos que consiste en el uso alterno del procesador y la espera de operaciones de E/S; se suele decir que se alternan ráfagas de CPU y de E/S. El ***buffering*** es un método que permite simultanear las operaciones de la CPU y de E/S para un mismo trabajo en una misma máquina.

Consiste en dotar a los dispositivos de E/S de *buffers* donde almacenar la información temporalmente. De esta forma, mientras la CPU está procesando un dato que se acaba de leer, el dispositivo puede estar realizando otra operación de entrada, situando el nuevo dato en el *buffer*. Análogamente, cuando se trata de una operación de salida, la CPU puede estar introduciendo los datos en el *buffer*, mientras el dispositivo realiza una operación de salida. En la figura 1.5 podemos ver un diagrama temporal donde se compara un sistema sin *buffer* y otro con un *buffer*. Se puede apreciar cómo el sistema que posee el *buffer* puede solapar las operaciones de cálculo con el primer dato mientras realiza la entrada del segundo. Esto permite aumentar el rendimiento del sistema, ya que el sistema sin *buffer* sólo puede empezar a leer el segundo dato cuando termina de procesar el primero.

Los *buffers* permiten reducir las diferencias entre la velocidad a la que se procesan los datos y a la que se realizan las operaciones de E/S, consiguiendo aumentar el rendimiento del sistema al mantener menos tiempo ociosa a la CPU. Pero los *buffers* no son realmente efectivos si los trabajos no presentan un intercalamiento de operaciones de E/S y de cálculo. Así, los trabajos limitados por la CPU que necesitan ráfagas largas de CPU y realizan pocas

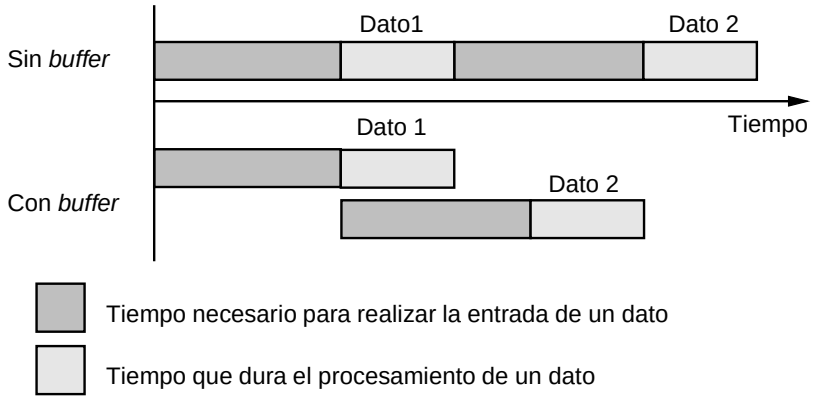

Figura 1.5: *Buffering*

operaciones de E/S mantendrán los *buffers* de entrada siempre llenos y los de salida vacíos. Por otro lado, los limitados por la E/S, que realizan muchas operaciones de E/S y necesitan ráfagas de CPU cortas, mantienen vacíos los *buffers* de entrada y llenos los de salida, por lo que la CPU se ve limitada por la velocidad de los dispositivos.

1.2.2.3. *Spooling*

La aparición de los discos introduce una nueva forma de procesamiento denominada ***spooling***[1] que permite solapar las operaciones de E/S y de la CPU utilizando una sola máquina. La diferencia fundamental entre los discos y las cintas (dispositivos de almacenamiento utilizado hasta el momento) es su modo de acceso. Los discos son dispositivos de acceso aleatorio, por lo que se puede pasar rápidamente de una zona a otra, mientras que las cintas sólo permiten un acceso secuencial.

En un sistema de *spooling* (figura 1.6), las tarjetas son leídas directamente del lector de tarjetas al disco. Cuando se ejecuta un trabajo, el sistema operativo satisface sus peticiones de entrada del lector de tarjetas leyendo del disco. De forma similar, cuando el trabajo hace una petición para escribir en la impresora una línea, la información se copia en un *buffer* del sistema situado en el disco. Cuando el trabajo se completa, la salida se envía a la impresora.

[1] El nombre es un acrónimo de *simultaneous peripheral operation on-line*.

El *spooling*, en esencia, usa el disco como un gran *buffer*, para adelantar la lectura tanto como sea posible en dispositivos de entrada y para almacenar la información de salida hasta que los dispositivos sean capaces de aceptarla.

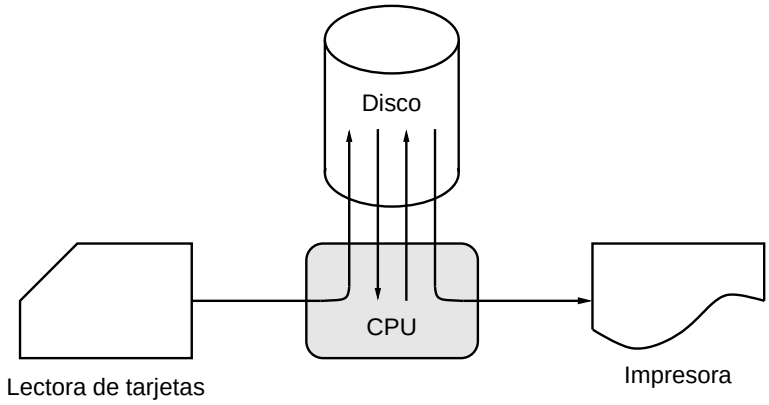

Figura 1.6: *Spooling*

El *spooling* solapa la E/S de un trabajo con la computación de otros. Incluso en un sistema simple el *spooler* puede estar leyendo la entrada de un trabajo mientras está imprimiendo la salida de otro diferente.

Esta técnica tiene un beneficio directo sobre el rendimiento del sistema. Por el costo de algo de espacio en disco, la CPU puede solapar la computación de un trabajo con la E/S de otros. Así, el *spooling* puede mantener trabajando tanto a la CPU como a los dispositivos de E/S a velocidades mucho más altas.

El *spooling* permite disponer de un ***pool* de trabajos**, es decir, un conjunto de trabajos en disco dispuestos para ejecutarse. Esto va a permitir al sistema operativo seleccionar el próximo trabajo a ejecutar cuando el que está ejecutando actualmente tiene que realizar una operación de E/S. Cuando los trabajos están en tarjetas o cinta magnética, no es posible saltar y ejecutarlos en un orden diferente, sino que tienen que ser ejecutados secuencialmente. Sin embargo, cuando varios trabajos están en un dispositivo de acceso directo, tal como un disco, es posible hacer una **planificación** de éstos.

1.2.3. Sistemas por lotes multiprogramados

La operación fuera de línea, el *buffering* y el *spooling* tienen sus limitaciones para solapar la E/S. Un solo trabajo no puede, en general, mantener

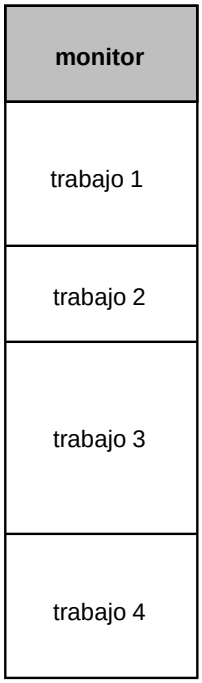

Figura 1.7: Estado de la memoria en un sistema de multiprogramación

ocupados todo el tiempo la CPU y los dispositivos de E/S. Una forma de conseguirlo es que el sistema operativo mantenga varios trabajos en memoria a un tiempo como se muestra en la figura 1.7, esto es lo que se conoce como **multiprogramación**. Este conjunto de trabajos será normalmente un subconjunto de los que se mantienen en el disco (ya que la capacidad de la memoria es usualmente mucho menor que la del disco). El sistema operativo escoge uno de los trabajos que está en memoria y empieza a ejecutarlo. Si en algún momento el trabajo tiene que esperar algo, como por ejemplo que se monte una cinta, que se dé una orden, o que se complete una operación de E/S, el sistema operativo podrá elegir otro trabajo para ejecutarlo. En un sistema donde no existe la multiprogramación, la CPU estaría ociosa durante este tiempo. Con la multiprogramación aumenta la utilización de la CPU organizando los trabajos de forma que ésta siempre tenga algo que ejecutar. Esto es lo que se representa en la figura 1.8.

Un concepto básico en estos sistemas es el **grado de multiprogramación**, que se define como el número de trabajos cargados simultáneamente en memoria.

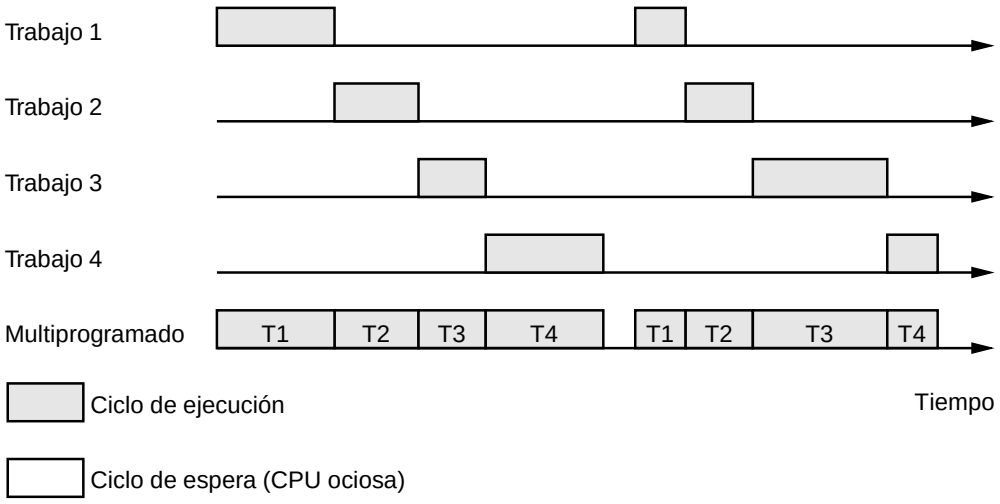

Figura 1.8: Ejecución de trabajos en un sistema multiprogramado

Los **sistemas operativos por lotes multiprogramados** son claramente más sofisticados que los anteriores, pues son los primeros que toman decisiones por los usuarios. Todos los trabajos que entran en el sistema se mantienen en el *pool* de trabajos a la espera de obtener espacio en la memoria principal para comenzar su ejecución. Si varios trabajos están listos para ser llevados a memoria, y no hay suficiente sitio para todos ellos, entonces el sistema debe elegir. Esta decisión es lo que se conoce como **planificación de trabajos**. Cuando el sistema operativo selecciona un trabajo del *pool*, lo carga en memoria para su ejecución. Cuando tenemos varios trabajos en memoria dispuestos para ejecutarse, también el sistema debe elegir entre ellos para decidir cuál es el que se ejecutará primero; esto es lo que se conoce como **planificación de la CPU**. Todos estos aspectos serán estudiados con mayor amplitud en el capítulo **??**.

El tener varios trabajos cargados en memoria al mismo tiempo requiere un manejo de la memoria más complejo que en los sistemas anteriores, al tener que decidir en qué zonas van a ubicarse los distintos trabajos. En el capítulo **??** se analizará este problema con mayor amplitud. Por otro lado, la existencia de varios trabajos activos simultáneamente hace imprescindible proteger los distintos recursos a los que pueden acceder, tales como la memoria, dispositivos de E/S y la CPU, con objeto de evitar interferencias entre ellos. Los mecanismos de protección necesarios se tratarán en más profundidad en el apartado 1.7.1.

1.2.4. Sistemas de tiempo compartido

Los sistemas por lotes con multiprogramación proporcionan un entorno en el que los diferentes recursos del sistema (CPU, memoria, dispositivos de E/S) se utilizan de una forma más efectiva. Sin embargo, desde el punto de vista del programador o del usuario, estos sistemas presentan algunas dificultades. Éstas son debidas a la falta de interacción entre los usuarios y el sistema, por ejemplo, los programas deben ser depurados estáticamente, a partir de los volcados de memoria. El programador no puede modificar un programa a medida que se ejecuta para estudiar su comportamiento.

Los **sistemas de tiempo compartido** son una extensión lógica de los sistemas de multiprogramación. En ellos, múltiples trabajos son ejecutados por la CPU intercambiándose entre ellos, además los cambios ocurren tan frecuentemente que los usuarios pueden interaccionar con cada programa mientras se está ejecutando. Un sistema de tiempo compartido es un sistema de multiprogramación interactivo; esto permite que el usuario dé instrucciones al sistema operativo o a un programa en ejecución directamente, y reciba una respuesta inmediata. De esta forma el usuario puede experimentar fácilmente y ver los resultados inmediatamente.

En estos sistemas, el usuario envía órdenes y espera los resultados, por tanto, el **tiempo de respuesta** debería ser lo más pequeño posible. Esta es una de las condiciones principales que se le debe imponer a un sistema interactivo.

Los primeros computadores eran sistemas interactivos, ya que el sistema completo estaba a disposición del usuario. Esto permitía al programador gran flexibilidad y libertad en la prueba y desarrollo de programas. Pero daba lugar a que la CPU estuviera ociosa durante períodos largos de tiempo. Debido al alto coste de estos primeros computadores, el tiempo de CPU ocioso no era deseable, por lo que se desarrollaron los sistemas por lotes. Éstos mejoraron la utilización del sistema para los propietarios de los computadores.

Los sistemas de tiempo compartido son el resultado de intentar proporcionar un uso interactivo del computador a un costo razonable. Un sistema operativo de tiempo compartido utiliza la planificación de la CPU y la multiprogramación para proporcionar a cada usuario una pequeña porción de un computador de tiempo compartido. Cuando un programa se ejecuta, normalmente lo hace durante períodos cortos de tiempo, teniendo que liberar la CPU cada vez que transcurre el tiempo máximo permitido o bien cuando necesita realizar una operación de E/S. Siempre que se dan estas circunstancias

el sistema pasa a ejecutar otro programa diferente.

Un sistema operativo de tiempo compartido permite a muchos usuarios utilizar simultáneamente el computador; se trata, por tanto, de un sistema **multiusuario**, al contrario que los sistemas anteriores que eran todos monousuario. Esto se consigue dando a cada usuario el uso de la CPU durante un intervalo corto de tiempo. Es decir, la CPU va atendiendo a los distintos usuarios del sistema durante períodos cortos de tiempo, dándoles a estos la impresión de que el computador está a su entera disposición.

La tabla 1.1 muestra las diferencias entre un sistema de multiprogramación por lote y uno de tiempo compartido, donde se pone de manifiesto la diferencia básica entre ambos sistemas: el primero intenta maximizar el rendimiento del sistema, mientras que el segundo ofrece un entorno interactivo a los usuarios, con un tiempo de respuesta razonable.

	Multiprogramación por lotes	Tiempo compartido
Objetivo principal	Maximizar el uso del procesador	Minimizar el tiempo de respuesta
Origen de las instrucciones al sistema operativo	Se proporcionan las instrucciones en un lenguaje de control de trabajos junto al trabajo	Órdenes introducidas desde una terminal

Cuadro 1.1: Comparativa entre un sistema multiprogramado y un sistema de tiempo compartido

La idea del tiempo compartido fue demostrada al principio de los años 60, pero como estos sistemas eran más difíciles y caros de construir, no llegaron a ser comunes hasta principios de los años 70.

El tiempo compartido plantea nuevos problemas para el sistema operativo. La presencia de varios usuarios interactivos introduce la necesidad de proteger los ficheros para que sólo los usuarios autorizados puedan tener acceso a éstos.

1.3. Sistemas para computadores personales

Con el tiempo los costos del hardware han disminuido mucho, haciendo posible de nuevo tener un computador dedicado a un solo usuario. Éstos se conocen comúnmente como **computadores personales**.

Hasta hace poco, los procesadores de estos computadores habían perdido las características que se necesitaban para proteger al sistema operativo de los programas de los usuarios. Los sistemas operativos que se escribían para ellos eran monousuarios y monoprogramados. Las metas de estos sistemas operativos también son diferentes, en vez de intentar maximizar la utilización de la CPU y de los periféricos, optan por la comodidad del usuario.

A medida que los computadores personales se han ido haciendo más potentes, sus sistemas operativos han adoptado algunas de las características de los sistemas estudiados anteriormente.

1.4. Sistemas de tiempo real

Los **sistemas operativos de tiempo real** son sistemas de propósito especial construidos para resolver problemas concretos. Se utilizan cuando existen requisitos estrictos de tiempo en la operación del procesador o en el flujo de datos. En estos sistemas la corrección de su operación depende no sólo de los resultados lógicos de la computación, sino también del tiempo que tarda en producirse estos resultados. Algunos ejemplos de uso de sistemas de tiempo real son el control de procesos de producción en la manufacturación de productos, los sistemas de inyección de gasolina de algunos automóviles, etc. En todos los casos, unos sensores suministran información sobre el entorno y el sistema tiene que responder de forma adecuada a los cambios que se produzcan. Esta respuesta tiene unos límites bien definidos de tiempo, y si no se produce dentro de éstos el sistema fallará.

Se pueden distinguir dos tipos de sistemas de tiempo real: los duros y los suaves. Para los sistemas de tiempo real duros es absolutamente imprescindible que las respuestas se produzcan dentro de los límites de tiempo especificados. Los sistemas de tiempo real suaves son menos restrictivos, sigue siendo importante que los tiempos de respuesta estén dentro de los límites, pero si alguna vez no lo están, el sistema seguirá funcionando. Estos sistemas tienen una utilidad más limitada que los duros, ya que no pueden ser utilizados para control industrial y robótica. Sin embargo, hay varias áreas donde son útiles,

por ejemplo, multimedia, realidad virtual, etc. Estos sistemas disponen de características de los sistemas operativos avanzados que no están disponibles en los sistemas de tiempo real duros.

1.5. Sistemas con más de un procesador

Todos los sistemas considerados hasta ahora se ejecutan en computadores con un solo procesador. Posteriormente, ha empezado a construirse sistemas con más de un procesador, por lo que el sistema operativo se ha tenido que adaptar a esta circunstancia. Estos sistemas proporcionan un ahorro de dinero con respecto a los sistemas de un solo procesador, debido a la compartición entre todos ellos de los mismos periféricos, etc. Existen distintos tipos de sistemas con múltiples procesadores: los sistemas paralelos, en red y distribuidos.

1.5.1. Sistemas paralelos

Estos sistemas tienen varios procesadores que comparten el bus del sistema, el reloj, la memoria, etc. Hay varias razones que justifican la construcción de este tipo de sistemas. Una ventaja es el aumento de rendimiento. Es de esperar que al aumentar el número de procesadores que trabajan juntos, aumente la cantidad de trabajo que se realiza por unidad de tiempo. Esto es lo que suele ocurrir, aunque el aumento de rendimiento no es exactamente proporcional al número de procesadores, sino algo menor, ya que suele haber cierta sobrecarga en este tipo de sistemas para hacer que todos los procesadores trabajen de forma coordinada.

Otra razón para la construcción de este tipo de sistemas es el aumento de fiabilidad. En ellos, el fallo de un procesador no paraliza el sistema completo, sólo se hace más lento.

La mayor parte de estos sistemas usan hoy día el **multiprocesamiento simétrico**, en el cual cada procesador ejecuta una copia idéntica del sistema operativo. Algunos sistemas usan lo que se conoce como **multiprocesamiento asimétrico**, en el cual a cada procesador se le asigna una tarea específica. Un procesador maestro controla el sistema; los demás procesadores pueden esperar a recibir instrucciones del procesador maestro o tienen tareas predefinidas.

1.5.2. Sistemas en red y distribuidos

A mediados de la década de los 80 se empezaron a construir redes de ordenadores personales que ejecutaban sistemas operativos en red y distribuidos.

En los dos casos tenemos un conjunto de ordenadores conectados mediante una red de comunicación que establece la necesidad de un protocolo de comunicaciones entre los distintos equipos. Una diferencia entre ambos es el tipo de sistema operativo que ejecutan. En el caso de los sistemas en red cada máquina tiene su propio sistema operativo, no teniendo que ser iguales los de todas las máquinas; sobre éste se ejecuta un software de red que sirve para comunicar las máquinas. Los sistemas distribuidos se caracterizan porque el sistema operativo que se ejecuta en todas las máquinas es idéntico.[2]

Existe otra diferencia fundamental entre ambos tipos de sistemas, en el caso de un sistema operativo en red los usuarios son conscientes de la existencia de múltiples computadores, pueden acceder a máquinas remotas o copiar ficheros de una máquina a otra, pero saben que están accediendo a un recurso remoto. Los sistemas operativos en red no son fundamentalmente diferentes de los sistema operativos para equipos con un solo procesador. Tienen características adicionales para permitir la comunicación, pero éstas no modifican la estructura esencial del sistema operativo.

Sin embargo, los sistemas distribuidos aparecen ante los usuarios como si se tratara de un sistema uniprocesador tradicional. La distribución de los recursos es transparente a los usuarios, ya que no son conscientes de qué procesador ejecuta su programa, dónde están localizados los ficheros, etc; todo esto es manejado de forma automática por el sistema operativo. Una de las formas de conseguir esta transparencia es a través del **modelo cliente-servidor**, que permite describir la interacción de los programas en un sistema distribuido. De acuerdo a este modelo, cualquier programa que se ejecute en el sistema puede proveer servicios para otros, o pedirlos. Los programas de usuario que solicitan servicios son los denominados **clientes**, y aquellos que proveen los servicios se denominan **servidores**.

[2] Actualmente se construyen sistemas distribuidos en los que cada máquina puede ejecutar un sistema operativo diferente. En éstos existe una capa de software denominada *middleware* que es la que proporciona las características distribuidas del sistema.

1.6. Clasificación de los sistemas operativos

En los apartados anteriores hemos visto cómo han evolucionado los sistemas operativos con el objetivo de conseguir siempre un mayor rendimiento del sistema. Esto ha permitido adaptar el sistema operativo a nuestras necesidades. Así, los sistemas operativos se pueden clasificar en base a distintos criterios, como muestra la figura 1.9. Esta clasificación no es exclusiva, es decir, un sistema operativo monousuario puede ser interactivo y multiprogramado.

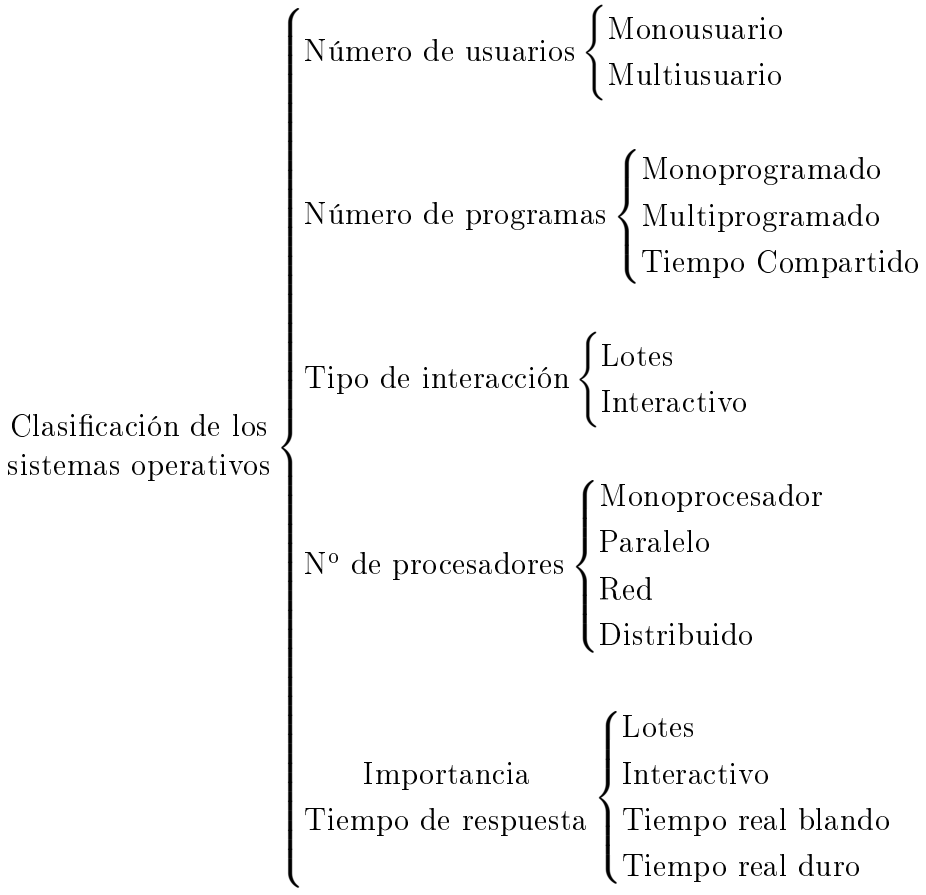

Figura 1.9: Clasificación de los sistemas operativos

1.7. Aspectos hardware

1.7.1. Mecanismos hardware de protección

Al ir aumentando la complejidad de los sistemas operativos también lo hace la necesidad de protección de recursos tales como la memoria, la CPU y los dispositivos de E/S. Por ejemplo, cuando tenemos en memoria varios procesos junto con el sistema operativo debemos evitar que un proceso acceda a la zona de memoria donde está cargado el sistema operativo u otro proceso.

Una forma de conseguir la protección de todos estos recursos consiste en distinguir dos tipos de instrucciones hardware, aquellas que no presentan ningún tipo de problema de protección y las que sí son conflictivas desde el punto de vista de la protección de los recursos. Para poder hacer esta distinción es necesario disponer de dos modos de ejecución diferentes, un modo privilegiado que permite acceder a cualquier instrucción hardware y un modo normal que sólo permite acceder al subconjunto de instrucciones que no son conflictivas (**modo de operación dual**). Pero para poder diferenciar entre dos modos de ejecución diferentes es necesario disponer de un **bit de modo**, que dependiendo de su valor establecerá un modo de ejecución u otro.

Por tanto, aquellas arquitecturas que cuentan con el bit de modo permiten establecer un modo de ejecución privilegiado que se suele denominar modo supervisor o modo núcleo que permite ejecutar cualquier instrucción, y un modo normal o modo usuario que sólo permite acceder a las instrucciones no privilegiadas. Si se intenta ejecutar una instrucción privilegiada en modo usuario, el hardware la tratará como una instrucción no válida produciendo una excepción[3] y, por tanto, pasará el control al sistema operativo.

A continuación se describe la forma en que se protegen los diferentes recursos del sistema.

Las instrucciones de E/S son instrucciones protegidas, por lo que un proceso que se ejecute en modo usuario no puede realizar su propia E/S, sino que es el sistema operativo el que la tendrá que realizar en su nombre. Esta es la forma en la que se protegen los dispositivos de E/S.

La protección de la CPU pretende evitar que un proceso tome el control de la CPU y no lo devuelva. Para ello se suele utilizar un temporizador que provoca interrupciones cada cierto tiempo, permitiendo que el sistema operativo tome el control. En este caso, las instrucciones que permiten modificar

[3]Una excepción o trampa es una interrupción producida por el software.

los parámetros del temporizador deberían ser privilegiadas.

En el caso de la memoria, el espacio de direcciones que ocupa cada proceso se suele proteger mediante el uso de dos registros que establecen las direcciones base y límite de este espacio de direcciones. Cada vez que un proceso hace una referencia a memoria, se compara la dirección a la que quiere acceder con los valores de estos registros, para comprobar si la dirección es o no válida. En este caso, las instrucciones que permiten modificar los valores de estos registros se definen como privilegiadas.

1.7.2. Técnicas para la realización de operaciones de E/S

Una operación de E/S consiste en la transferencia de información desde la memoria a un dispositivo (operación de salida), o bien desde un dispositivo a la memoria (operación de entrada).

Las operaciones de E/S se pueden llevar a cabo de distintas formas, dependiendo de las características del dispositivo del que se disponga. La técnica que se utilice para realizar estas operaciones va a tener un efecto importante sobre el rendimiento del sistema de computación. Estas técnicas son las siguientes:

- E/S controlada por programa
- E/S controlada por interrupciones
- Acceso directo a memoria (DMA)

1.7.2.1. E/S controlada por programa

Cuando el procesador está ejecutando un programa y encuentra una instrucción de E/S, ejecuta esta instrucción enviando una orden al controlador del dispositivo implicado. Cuando se emplea E/S programada, también denominada **escrutación**, la CPU se encarga de controlar si la operación ha terminado o no comprobando periódicamente el estado del controlador (de ahí el nombre de escrutación). En este caso el controlador no realiza ningún tipo de acción para avisar al procesador de que la operación de E/S ha finalizado, o de que ha ocurrido un error, etc.; sino que es el propio procesador el que tiene que comprobarlo cada cierto tiempo.

Con esta técnica, el procesador es el responsable de extraer los datos de la memoria principal cuando va a realizarse una operación de salida o almacenar los datos en la memoria principal cuando se realiza una entrada. El software de E/S se escribe de manera que el procesador ejecute unas instrucciones que le otorguen el control directo sobre la operación de E/S, incluyendo la comprobación del estado de los dispositivos, el envío de órdenes de lectura o escritura y la transferencia de los datos.

La figura muestra un ejemplo de utilización de la E/S programada para leer datos desde un dispositivo externo (por ejemplo, un registro de una cinta) hacia la memoria. Los datos se leen en unidades de una palabra (por ejemplo, 16 bits) cada vez. Por cada palabra que se lea, el procesador debe permanecer en un bucle de comprobación del estado hasta que determine que la palabra está disponible y puede ser escrita en memoria.

1.7.2.2. E/S controlada por interrupciones

Cuando el procesador está ejecutando un programa y encuentra una instrucción de E/S, ejecuta esta instrucción enviando una orden al controlador del dispositivo implicado. Cuando se emplea E/S controlada por interrupciones, la CPU se desentiende de la operación de E/S mientras esta se realiza, pudiendo dedicarse mientras la operación está en curso a ejecutar otro programa diferente.

Cuando se utiliza esta técnica, el controlador avisa al procesador mediante una interrupción cuando tiene los datos disponibles para su transferencia. El procesador abandonará la tarea que estaba realizando para atender la interrupción y se encargará de la transferencia de los datos y podrá continuar con lo que estaba haciendo.

Vamos a explicar cómo funciona esta técnica desde el punto de vista del dispositivo de E/S. En un operación de entrada, el controlador del dispositivo recibe la orden LEER desde el procesador. El controlador procede a realizar la lectura de los datos desde el periférico asociado. Cuando los datos están en el registro de datos del controlador, éste envía una señal de interrupción al procesador a través de una línea de control. El controlador se queda esperando a que los datos le sean solicitados por el procesador. Cuando llegue esta solicitud, el controlador situará los datos en el bus de datos y estará listo para otra operación de E/S.

Desde el punto de vista del procesador, las acciones a realizar para una operación de entrada son las siguientes: el procesador envía una orden LEER,

guarda el contexto del programa en curso y se dedica a hacer otra cosa (por ejemplo, a ejecutar otro programa). Al finalizar cada ciclo de instrucción, el procesador comprueba si hubo una interrupción. Cuando se produce una, el procesador guarda el contexto del programa que se está ejecutando en ese momento y atiende la interrupción. En este caso, el procesador lee la palabra de datos del controlador de E/S y la almacena en la memoria. A continuación restaura el contexto del programa que interrumpió anteriormente para atender la interrupción, u otro diferente, para seguir con su ejecución.

1.7.2.3. Acceso directo a memoria

Cuando se tienen que mover grandes volúmenes de datos entre los dispositivos de E/S y la memoria, se necesita una técnica eficiente que lo haga de forma rápida. Esta técnica para realizar operaciones de E/S se denomina **acceso directo a memoria**.

Esta técnica funciona de la siguiente forma: cuando el procesador desea leer o escribir un bloque de datos, emite una orden hacia la controladora DMA, enviándole la siguiente información:

- Tipo de operación que desea realizar: lectura o escritura.
- La dirección del dispositivo de E/S involucrado.
- La dirección inicial de memoria desde la que se va a leer o en la que se va a escribir.
- Número de palabras a leer o escribir.

A partir de este momento, el procesador puede continuar con otro trabajo, ya que ha delegado la operación de E/S en el controlador DMA. Éste se encarga de transferir el bloque completo, palabra a palabra, hacia la memoria o desde ella, sin la intervención del procesador. Cuando se completa la transferencia, el controlador DMA envía una señal de interrupción al procesador. Por tanto, el procesador sólo se ocupa de la operación de E/S al principio y al final de ésta.

1.8. Resumen

Dentro de un sistema de computación, uno de los elementos más importantes es el sistema operativo. El objetivo de éste es abstraer los aspectos complejos del sistema ofreciendo una máquina virtual, donde el usuario puede trabajar en un entorno más cómodo, sin necesidad de conocer el funcionamiento del hardware.

Los sistemas operativos actuales han heredado características de todos los que han surgido a lo largo de la historia: por lotes, multiprogramados, de tiempo compartido, etc. Esto se puede ver en la figura 1.10.

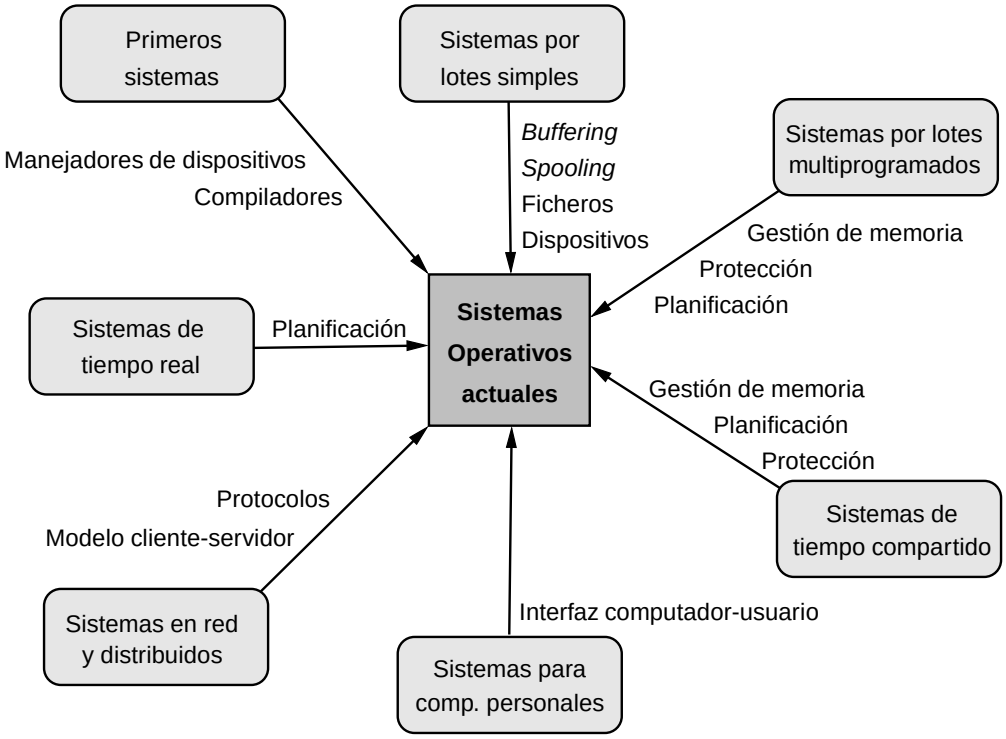

Figura 1.10: Evolución de los sistemas operativos

Uno de los primeros aspectos que incorporaron los sistemas operativos fueron los manejadores de dispositivos. Con ellos se consigue uno de los objetivos, abstraer la capa hardware con el fin de hacer el sistema de computación más fácil de manejar.

Los sistemas por lotes simples incorporan dos técnicas, el *buffering* y el

spooling, cuyo objetivo es solapar las operaciones de la CPU con las de E/S, con el fin de aumentar el rendimiento del sistema y mantener a la CPU inactiva el menor tiempo posible. Además, con la aparición de los discos surgen los ficheros y los denominados dispositivos lógicos, que permiten abstraer aún más el hardware.

Sin embargo, el gran avance se va a producir en los sistemas por lotes multiprogramados, que consiguen mantener en memoria varios trabajos a la vez, con la posibilidad de intercalar sus ejecuciones cuando tengan que esperar. Esto introduce una serie de cambios en el sistema operativo, debido a la necesidad de administrar la memoria para alojar los distintos trabajos, planificar los distintos recursos del sistema (incluida la CPU) entre todos los programas que se están ejecutando.

Con la aparición de los sistemas operativos de tiempo compartido y las terminales aparecen los primeros sistemas multiusuario, en los que varios usuarios pueden trabajar interactivamente con una máquina.

El descenso en el precio del hardware permitió la construcción de ordenadores personales. Los sistemas operativos para estos ordenadores han aprovechado las características de los sistemas construidos para los grandes computadores.

Los sistemas de tiempo real se pueden clasificar en dos tipos: duros y suaves. Los primeros se utilizan como dispositivos de control en aplicaciones dedicadas, y tienen unos requisitos de tiempo bien definidos, que deben cumplirse ya que de otro modo el sistema fallará. Los sistemas de tiempo real suaves son menos restrictivos en cuanto a estas limitaciones de tiempo.

Los sistemas paralelos tienen más de un procesador que comparten el bus, la memoria y diversos dispositivos. Estos sistemas proporcionan un aumento del rendimiento y una mayor fiabilidad.

Los sistemas operativos en red permiten comunicar diversas máquinas pudiendo los usuarios de éstas utilizarlas de forma remota, acceder a sus recursos, etc. Los sistemas operativos distribuidos permiten hacer esto mismo pero de una forma transparente a los usuarios. Una forma de conseguirlo es mediante lo que se denomina el modelo cliente-servidor.