APUNTES DE

Introducción a los Sistemas Operativos

2º DE GRADO EN INGENIERÍA INFORMÁTICA

TEMA 1. INTRODUCCIÓN

CURSO 2021/2022

ÍNDICE GENERAL

CAPÍTULO 1

INTRODUCCIÓN

1.1 CONCEPTO DE SISTEMA OPERATIVO

Un sistema operativo es un componente software que, en la arquitectura de un sistema de computación actual, se encuentra justo sobre el hardware y por debajo de los programas del sistema y aplicaciones (programas de usuario), como se puede apreciar en la figura 1.1. De ahí que se pueda afirmar que un sistema operativo es, en cierto sentido, un intermediario entre los usuarios de esos programas y utilidades, y el hardware del ordenador.

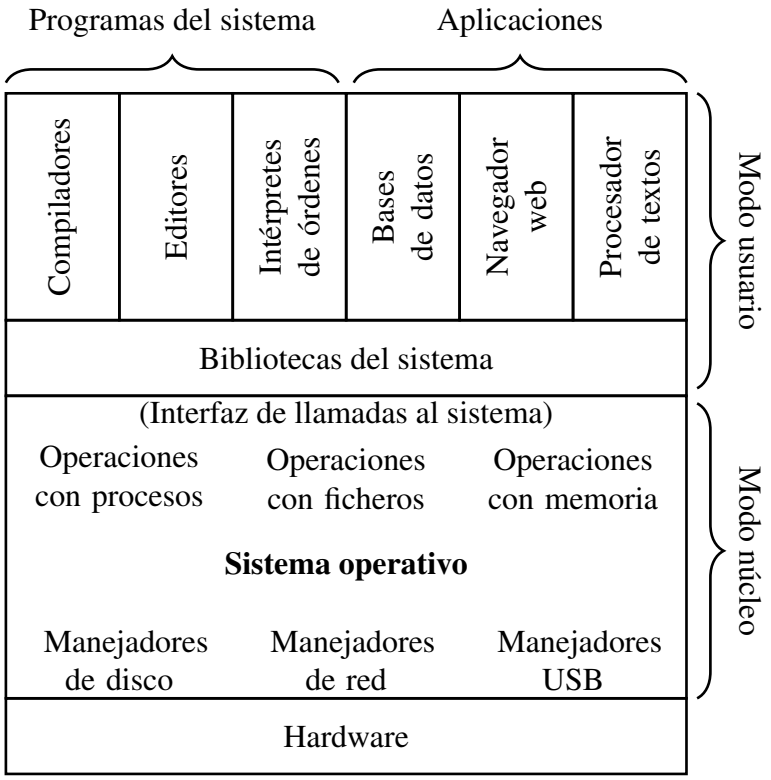


Figura 1.1: Perspectiva abstracta de los componentes de un sistema de computación.

Las funciones que desempeña un sistema operativo son tan complejas que no existe una única definición de sistema operativo. No obstante, de manera muy general, podemos ver un sistema operativo desde dos puntos de vista: como *máquina extendida o*

virtual, o como *controlador de recursos*.

Según el primer punto de vista, el sistema operativo es un programa que oculta el funcionamiento del hardware al programador y presenta una interfaz de la máquina sencilla y amigable. Entre otras cosas, el sistema operativo oculta los detalles relativos al manejo de interrupciones, relojes, control de memoria y otras características de bajo nivel. Al ocultar todos estos detalles, los programas que se ejecutan por encima del sistema operativo no ven el hardware real de la máquina sino la visión de la misma que les proporciona el sistema operativo.

Además, el sistema operativo añade nuevos componentes y funcionalidades que el hardware por sí mismo no proporciona, como el almacenamiento y organización de la información por medio de un sistema de ficheros, la capacidad de ejecutar varios programas a la misma vez, etc. Todas estas capacidades son ofrecidas por el sistema operativo a través de *llamadas al sistema*, como veremos más adelante.

Por tanto, según este primer punto de vista, el sistema operativo presenta a los programas que trabajan por encima de él una *nueva interfaz* de la máquina real que hace que la programación y el uso del computador sean más fáciles. De ahí que esta nueva interfaz se pueda entender como una máquina virtual o una máquina extendida.

Desde el segundo punto de vista, el sistema operativo es un programa que controla todas las piezas de un sistema complejo y proporciona una asignación ordenada y controlada de los procesadores, memoria, dispositivos de E/S, etc., entre los distintos programas que compiten por ellos.

Así, por ejemplo, si varios programas quieren utilizar la impresora a la vez, el sistema operativo debe leer la información que ha de ir a la impresora, guardar dicha información en varios ficheros (uno por trabajo de impresión) y, finalmente, ir enviando los ficheros uno a uno a la impresora. Si no lo hacemos así y dejamos que varios programas escriban directamente en la impresora, entonces la salida por la misma puede ser una mezcla de la información de todos ellos, ya que el control de la CPU se pasa rápidamente de un programa a otro (como también veremos después) y puede ocurrir que un programa no envíe toda su salida a la impresora antes de que se pase la CPU a otro programa que se ponga también a imprimir.

Lo mismo ocurre con otros recursos hardware como la memoria, los dispositivos de E/S, etc.; si hay varios usuarios[1] trabajando a la vez en el sistema, entonces se necesitará un control y protección de dichos recursos.

En resumen, según este segundo punto de vista, el sistema operativo es un controlador de recursos que debe llevar un registro de la utilización de los mismos, dar paso a las solicitudes de uso de los recursos que le llegan, llevar cuenta de ese uso, y mediar en los conflictos producidos por las solicitudes de distintos programas y usuarios.

Tras analizar los dos puntos de vista, podemos concluir que un sistema operativo no es nada mágico; no es más que un programa complejo que se ejecuta periódicamente para realizar ciertas tareas o cuando un programa solicita algún servicio.

Sistemas operativos hay muchos, cada uno con sus ventajas e inconvenientes en función de las tareas a realizar, entorno de ejecución, precio, etc. Entre los que existen actualmente, podemos destacar las distintas versiones de Microsoft Windows, de Apple macOS, de Linux, etc., sin olvidarnos de sistemas operativos como iOS y Android, presentes en muchos dispositivos móviles actuales.

[1] Evidentemente, un usuario no trabaja «directamente» en un sistema, sino que lo hace a través de la ejecución en ese sistema de uno o más programas que le pertenecen. Además, hay que tener en cuenta que un usuario no siempre se corresponde con una persona física; puede haber *usuarios del sistema* que son necesarios para el correcto funcionamiento del mismo. La definición exacta de qué es un usuario también puede cambiar de un sistema operativo a otro.

Una idea que debe quedar clara es que las características de un sistema operativo dependen muchas veces de las propiedades del hardware sobre el que se ejecuta. Por ejemplo, un sistema operativo no puede garantizar la protección entre zonas de memoria de dos programas diferentes si el hardware no dispone del mecanismo adecuado. Esto ha hecho que los sistemas operativos hayan evolucionado, por lo general, a la misma vez que el hardware sobre el que se ejecutan, como veremos en la sección 1.3.

1.2 PROTECCIÓN DEL SISTEMA OPERATIVO

Puesto que el funcionamiento de un sistema de computación depende, entre otras cosas, del correcto funcionamiento de su sistema operativo, es necesario que este esté protegido del resto de programas que se ejecutan en el computador.

La protección solo se consigue con la ayuda del hardware y, en concreto, de tres mecanismos básicos:

- **Modos de ejecución del procesador**. El procesador debe disponer de, al menos, dos modos de ejecución denominados *modo núcleo* (o *modo supervisor*) y *modo usuario*. El programa que se ejecuta cuando el procesador se encuentra en el primer modo es capaz de ejecutar cualquier instrucción disponible en el procesador. Este, por tanto, es el modo en el que se debe ejecutar el sistema operativo.

  En cambio, el programa que se ejecuta con el procesador en modo usuario no puede ejecutar ciertas instrucciones de la CPU, como las de E/S, de configuración de la memoria, etc.; si lo hace, provocará una *excepción*[2] que será tratada por el sistema operativo que hará que el programa termine su ejecución, mostrando, probablemente, un mensaje de error indicando que el programa ha sido abortado porque ha intentado ejecutar una instrucción privilegiada. En este modo deben ejecutarse todos los programas de usuario.

  Si las instrucciones de E/S son privilegiadas, ¿cómo realiza la E/S un programa de usuario? ¿Cómo puede, por ejemplo, leer de o escribir en un fichero que se encuentra en disco? La solución es que debe pedirle al sistema operativo que haga esa operación de E/S por él. Esta solicitud se realiza mediante una instrucción especial conocida como *llamada al sistema* (o *syscall*). Esta instrucción produce una acción similar a la de una interrupción que, como en el caso de las interrupciones, es tratada por el sistema operativo en modo núcleo. Hablaremos más de las llamadas al sistema en la sección 1.5.3.

- **Protección de la memoria**. El hardware debe tener algún mecanismo que impida que un programa acceda a zonas de memoria RAM que no le pertenecen. De esta forma, evitaremos que un programa pueda leer o modificar el código, los datos o la pila de otros programas y, sobre todo, del sistema operativo. Si no se protegiera la RAM, un programa podría modificar el vector de interrupciones[3], las rutinas de

[2] Una excepción es una condición de error que se produce durante la ejecución de un programa, por ejemplo, por una división por cero, la ejecución de una instrucción inexistente, un fallo de página, etc. Normalmente, las excepciones son tratadas por el sistema operativo, que realizará unas acciones u otras en función de la excepción producida.

[3] El vector de interrupciones es una estructura de datos en memoria RAM que contiene, para cada posible interrupción que se pueda ocurrir en la máquina, la dirección en memoria del procedimiento o *rutina de servicio de interrupción* del sistema operativo que debe tratarla. Esta estructura de datos es inicializada por el sistema operativo durante el arranque de la máquina. Cuando se produce una interrupción (es decir, una petición de atención por parte de algún dispositivo de E/S), el hardware obtiene del vector

servicio de interrupción del sistema operativo u otras partes del sistema operativo, y hacerse con el control de la máquina.

- **Interrupciones periódicas**. Para que un programa no se haga con el control de la CPU (por ejemplo, porque entre en un bucle infinito), es necesario que el hardware disponga de algún reloj o dispositivo similar que produzca interrupciones periódicas que permitan al sistema operativo hacerse con el control de la máquina cada cierto tiempo. Al tomar el control, el sistema operativo puede asignar la CPU a otro programa y así evitar que el sistema se bloquee.

Un aspecto importante es que la protección solo funcionará si los tres mecanismos mencionados existen y se usan de forma conjunta. Por ejemplo, si existen las interrupciones periódicas y la protección de memoria, pero el procesador no dispone de dos modos de funcionamiento, cualquier programa podrá cambiar la configuración tanto de la protección de memoria como de las interrupciones, haciendo que estas dejen de desempeñar su función.

1.3 HISTORIA Y EVOLUCIÓN

Los sistemas operativos han evolucionado a lo largo de los años. Dado que los sistemas operativos han estado relacionados históricamente con la arquitectura de los ordenadores en los que se ejecutan, analizaremos las generaciones sucesivas de ordenadores para ver cómo eran sus sistemas operativos. En esta evolución también vamos a ver cómo se introdujeron algunos conceptos que hoy en día se siguen utilizando.

1.3.1 Primera generación (1945–1955): válvulas y conexiones

Esta primera generación se caracteriza por la presencia de grandes máquinas que ocupaban habitaciones enteras. Estas máquinas estaban construidas mediante miles de válvulas de vacío y cables conectados a mano, consumían una gran cantidad de energía y su fiabilidad era muy pobre (la probabilidad de que se fundieran una o más válvulas de vacío era muy alta). Un solo grupo de personas diseñaba, construía, programaba, operaba y mantenía cada máquina. Se desconocían los lenguajes de programación y, por supuesto, los sistemas operativos.

Existía una interacción directa entre el programador y la máquina: cada programador reservaba la máquina a una determinada hora y la tenía por completo a su disposición. Los programas se introducían inicialmente instrucción a instrucción utilizando los conmutadores que existían en la consola, y más tarde mediante el uso de tarjetas perforadas.

Al reservar la máquina solo durante cierto tiempo, podían ocurrir dos cosas: que el programador terminara su trabajo en ese tiempo o que no lo terminara. En el primer caso se producían periodos de tiempo en los que la máquina no se utilizaba, lo cual no era bueno pues estas máquinas eran muy caras y había que aprovecharlas al máximo. En el segundo caso, el programador tenía que volver a reservar la máquina, por lo que el desarrollo de programas y ejecución de los mismos podía ser lento.

La programación se realizaba totalmente en lenguaje máquina y con frecuencia se utilizaban conexiones para controlar las funciones básicas del sistema. Además, el programador tenía que incluir en sus programas código para controlar los dispositivos de E/S que quería utilizar, en el caso de que existieran.

---

de interrupciones la dirección de la rutina adecuada y hace que la CPU cambie a modo núcleo y pase a ejecutarla. Dependiendo del hardware, el vector de interrupciones también puede contener las direcciones de los procedimientos que traten las excepciones y llamadas al sistema que se produzcan.

1.3.2 Segunda generación (1955–1965): transistores y sistemas de procesamiento por lotes

En esta generación surgieron diferentes desarrollos tecnológicos (lectores de tarjetas, impresoras de líneas, cintas magnéticas) y software (ensambladores, cargadores, enlazadores) que facilitaron la programación. La introducción del transistor fue fundamental, ya que hizo que los ordenadores se volvieran fiables, de forma que podían fabricarse y venderse a clientes. Por primera vez, se pudo establecer una clara separación entre los diseñadores, constructores, operadores, programadores y personal de mantenimiento.

Surgieron también las bibliotecas de funciones de E/S, por lo que el programador ya no tenía que desarrollar él mismo el código que debía controlar los dispositivos de E/S que quería utilizar, sino que ese código lo incorporaba a su programa desde esas bibliotecas.

La forma en la que se usaban las máquinas también cambió. Así, si se seguía permitiendo la interacción directa del programador con la máquina (como en la primera generación), este se tenía que hacer cargo de la colocación de las tarjetas perforadas, de la preparación de la impresora, de los cambios de cinta (por ejemplo, si quería compilar un programa en FORTRAN, colocaba la cinta con el compilador de ese lenguaje, si quería compilar un programa en COBOL, colocaba la cinta correspondiente), etc. Si el programador no tenía experiencia en la preparación de la máquina, se podían producir importantes pérdidas de tiempo.

Para evitar este problema de pérdida de tiempo, se adoptaron diversas medidas. Una fue contratar a un operador profesional (que actuaba como un *sistema operativo humano*) que era el que controlaba la máquina. En este caso, los programadores entregaban sus trabajos y debían esperar los resultados. Los programadores perdieron así la interacción directa con la máquina, interacción que no recuperarían hasta la siguiente generación.

Otra medida fue planificar los trabajos para evitar las pérdidas de tiempo por cambios de cinta. Así, los trabajos con requerimientos parecidos se agrupaban para ser procesados todos de una vez. Por ejemplo, se podían agrupar todos los programas fuentes en FORTRAN disponibles para después preparar la cinta con el compilador y compilarlos uno tras otro.

Pero, a pesar de todos esos cambios, aún persistían ciertos problemas. Por ejemplo, cuando se detenía un trabajo, el operador debía percatarse de la situación observando la consola, determinar por qué se detuvo el programa (terminación normal o anormal), efectuar un volcado de la memoria[4] si era necesario, cargar el lector de tarjetas o de cinta de papel con el siguiente trabajo y volver a preparar el ordenador.

Durante esta transición de un trabajo a otro, la CPU permanecía inactiva, algo inaceptable dado el alto coste de los equipos. Para superar este tiempo de inactividad, se desarrolló el sistema de *procesamiento por lotes*, lo que dio lugar a la creación de los primeros sistemas operativos rudimentarios, como el *monitor residente*. Este sistema operativo interpretaba las instrucciones de las tarjetas de control insertadas entre las tarjetas normales de datos. Dichas tarjetas de control indicaban el inicio y final de un trabajo, el tipo de trabajo a realizar (compilación de un programa fuente, ejecución de un programa, ...), etc. Se puede decir que estas tarjetas de control eran como las órdenes que hoy se introducen por teclado en una consola y que la impresora era como el monitor actual. En la figura 1.2 podemos apreciar la estructura de un trabajo por lotes en estos sistemas.

[4] Un *volcado de memoria* o *core dump* consiste en obtener una copia de la información disponible en memoria principal en un cierto instante, por ejemplo, cuando un programa falla. Esta copia se puede guardar en un fichero, se puede imprimir, etc. para analizar posteriormente las causas del fallo.

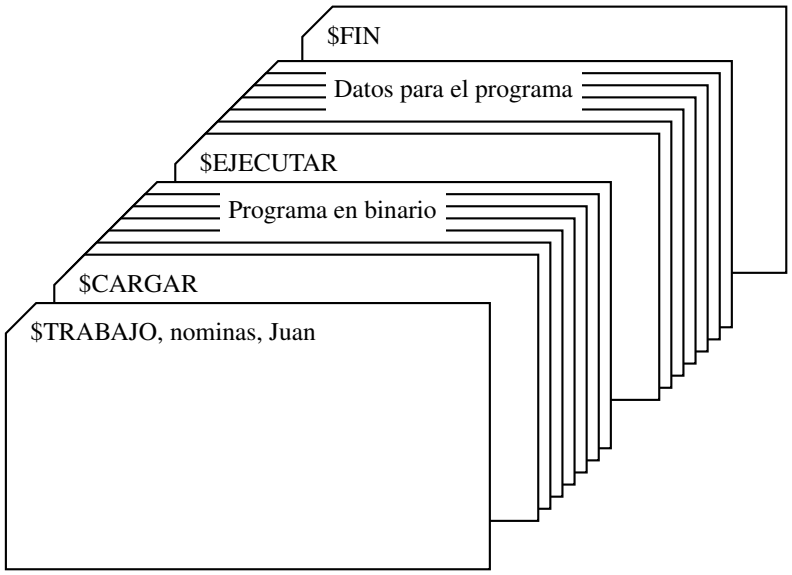


Figura 1.2: Estructura de un trabajo por lotes.

El monitor residente ocupaba de forma permanente una parte de la memoria principal, dejando el resto libre para la ejecución de programas. Este sistema operativo primitivo se componía de:

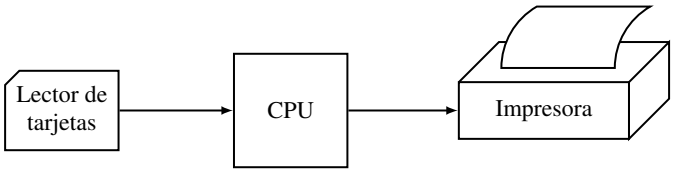
- Un intérprete de tarjetas de control.
- Un cargador de programas y aplicaciones.
- Manejadores de dispositivos de E/S[5] que, además, eran compartidos con los programas, por lo que los programadores no tenían que incluir dentro de sus propios programas el control de los dispositivos de E/S, etc.

Otro concepto de esta generación fue la *operación fuera de línea*, que trataba de solucionar el siguiente problema. Los dispositivos de E/S eran (y son) lentos en comparación con la velocidad de procesamiento de la CPU. Además, esta debía esperar a que finalizara cada operación de E/S que se realizaba. Esta espera hacía que hubiera periodos en los que la CPU se encontraba ociosa y, por tanto, infrautilizada (algo poco deseable, ya que, como hemos dicho, era un recurso muy caro al que había que sacarle el máximo provecho).

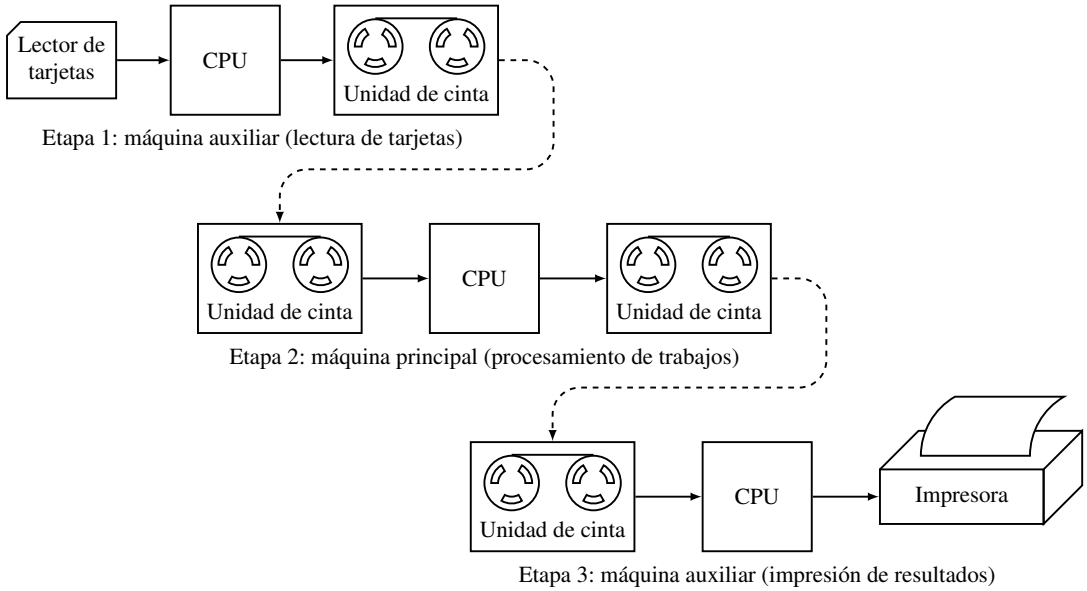
Una solución al problema descrito fue pasar de *operaciones en línea* (*on-line*), donde la CPU interactuaba directamente con los dispositivos de E/S lentos, como el lector de tarjetas o la impresora de líneas, a *operaciones fuera de línea* (*off-line*), donde la CPU interactuaba con dispositivos de E/S más rápidos, como las cintas (ver figura 1.3). En la operación fuera de línea podemos distinguir tres etapas:

- *1ª etapa*: los datos se leen de las tarjetas y se guardan en una cinta, la cual se debe llenar.

[5]Un manejador de dispositivo es una porción de código que forma parte del sistema operativo (al menos, desde un punto de vista lógico). Un manejador se ejecuta (en modo núcleo) cuando el sistema operativo necesita realizar alguna operación sobre el dispositivo correspondiente. Esto es así porque el manejador de un dispositivo es el único que conoce las peculiaridades de funcionamiento del mismo. Comúnmente, a los manejadores de dispositivo se les llama *drivers*, aunque en Linux también se les conoce como *módulos*. Otro posible nombre para un manejador que podemos encontrar en la bibliografía es el de *controlador*.



(a) Operación en línea.



(b) Operación fuera de línea.

Figura 1.3: Operación en línea (a) y fuera de línea (b) de dispositivos de E/S. En la operación fuera de línea, las etapas 1 y 3 son realizadas por máquinas relativamente baratas con CPU poco potentes. En cambio, la etapa 2 es realizada por una máquina equipada con una CPU cara, pero potente, a la que hay que sacar el máximo provecho. Como se observa, las cintas de salida de una etapa pasan a la entrada de la siguiente etapa.

- *2ª etapa*: los datos de la cinta se procesan y los resultados se guardan en otra cinta.
- *3ª etapa*: los datos de la cinta de salida se imprimen, se perforan en tarjetas, etc.

Lo que estamos haciendo es que los datos se procesen desde las cintas, cuya tasa de transferencia es mucho mayor que la que proporciona el lector de tarjetas, como se puede ver en la segunda etapa. Este esquema es útil cuando hay varias máquinas que desarrollan los trabajos de la 1ª y 3ª etapa, ya que, de lo contrario, la máquina principal puede quedar ociosa (esperando, por ejemplo, a que la máquina de la primera etapa termine de llenar la cinta que ha de procesar).

La operación fuera de línea dio lugar a una cuestión importante: si los programas pasan de usar tarjetas a usar cintas, entonces no deberían preocuparse del origen y destino de los datos, es decir, no deberían conocer si los datos se leen de una tarjeta o de una cinta, o si se escriben en una impresora o en una cinta. Esto hizo que surgiera la idea de *independencia de dispositivo de E/S*: los programas se escriben para usar dispositivos lógicos de E/S creados por el sistema operativo, y es este el que hace corresponder los dispositivos lógicos (por ejemplo, el dispositivo «almacén de tarjetas») con los físicos (el lector de tarjetas, una cinta, un disco, etc.). El acceso a los dispositivos lógicos siem-

pre se hace de la misma manera (mismas operaciones, mismas abstracciones de datos, etc.) independientemente del dispositivo físico que haya por debajo.

La operación fuera de línea hizo que se incrementara todavía más el tiempo que un programador debía esperar para obtener los resultados de sus trabajos, ya que, entre otras cosas, las cintas se tenían que llenar en la primera etapa y, una vez procesadas en la segunda etapa, los resultados obtenidos debían ser impresos (por ejemplo) en la tercera etapa.

Con la operación fuera de línea, se consiguió que la E/S y la CPU funcionasen en paralelo, ya que, mientras la CPU procesaba una cinta en la segunda etapa, la primera etapa leía tarjetas y la tercera imprimía resultados. El coste de este paralelismo era, sin embargo, bastante alto, ya que requería disponer de varias máquinas. Se debía, por tanto, conseguir un paralelismo parecido en una sola máquina. La solución la proporcionaron los *buffers* y los *spoolers*.

Un buffer es una zona de memoria principal de un cierto tamaño que se utiliza para el almacenamiento temporal de datos. Con los buffers, en lugar de que los datos pasen *directamente* del dispositivo de E/S a la CPU o viceversa, como se había hecho hasta ahora, los datos pasan a un buffer intermedio en memoria principal desde donde lee o en el que escribe la CPU. La idea, en el caso de la entrada desde un lector de tarjetas, es bastante sencilla: cuando la CPU está lista para trabajar con los datos disponibles en el buffer, se ordena al dispositivo que inicie de inmediato la siguiente lectura, tras lo cual la CPU se pone a procesar la información del buffer. En ese momento, tanto la CPU como el dispositivo se encuentran ocupados: la CPU procesando los datos de la tarjeta que se acaba de leer y el dispositivo leyendo la siguiente tarjeta. Con un poco de suerte, cuando la CPU esté lista para procesar nuevos datos, el dispositivo de entrada habrá terminado de leerlos y depositarlos en el buffer. La CPU puede entonces procesar los datos recién leídos, mientras el dispositivo comienza a leer los siguientes datos. De esta manera conseguimos que *la E/S de un programa se realice de forma simultánea (es decir, se solape en el tiempo) con el cómputo del mismo programa*.

Es importante darse cuenta de que la CPU no puede intervenir en la E/S del dispositivo, que tiene que ser realizada por este. Como mucho, la CPU puede transferir los datos entre el buffer de memoria principal y el dispositivo (suponiendo que este tiene algún tipo de memoria), aunque lo ideal es que esta transferencia se haga también sin la intervención de la CPU (por ejemplo, a través de DMA[6]).

El uso de buffers es útil si los promedios de procesamiento de datos (en bytes por segundo) de la CPU y el dispositivo de E/S son parecidos. En cambio, es poco útil si, por ejemplo, la CPU es mucho más rápida procesando datos que el dispositivo leyéndolos, pues la CPU estará la mayor parte de su tiempo ociosa, esperando a que el dispositivo le sirva datos. Obviamente, estamos suponiendo que solo hay un único programa en ejecución, por lo que no se puede pensar en darle la CPU a otro programa que puede estar listo para ejecutarse.

El manejo de buffers es generalmente una función del sistema operativo. El monitor residente, o los manejadores de dispositivos, incluyen buffers del sistema para cada dispositivo de E/S. Las operaciones de E/S (lecturas o escrituras) efectuadas por las aplicaciones solo producen una transferencia de datos desde o hacia estos buffers.

Por su parte, el mecanismo de *spooler* (*simultaneous peripheral operation on-line*, operación simultánea de periféricos en línea) permite la *superposición de la E/S de unos programas con el cómputo de otro*, obteniendo un beneficio similar al de las operaciones

[6]Recordemos que el DMA es un mecanismo hardware que permite las transferencias directas entre la memoria del dispositivo y la memoria principal sin la intervención de la CPU. Cuando la transferencia termina, el DMA avisa a la CPU de ello con una interrupción.

fuera de línea (ver figura 1.4). Este mecanismo se puede aplicar cuando se dispone de almacenamiento en disco o de cualquier otro tipo que permita un acceso aleatorio a cualquier parte de la información dentro del almacenamiento.

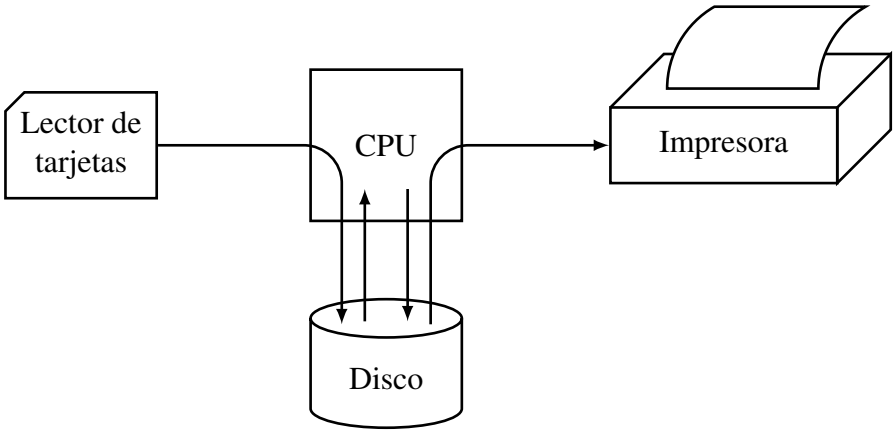


Figura 1.4: Mecanismo de *spooler*. Los datos de entrada y salida se almacenan temporalmente en el disco.

Veamos el funcionamiento en el caso de la entrada suponiendo que esta se produce desde un lector de tarjetas. Imaginemos que tenemos un programa en memoria que se está ejecutando. Durante su ejecución, la CPU[7] solicita al lector de tarjetas la siguiente tarjeta y, mientras el lector hace su trabajo, la CPU sigue ejecutando el programa que se encuentra en memoria (de nuevo, es evidente que el funcionamiento del dispositivo de E/S, el lector en este caso, no puede ser dirigido por la CPU). Cuando el lector de tarjetas tiene los datos de la tarjeta, avisa a la CPU, que interrumpe lo que está haciendo, pasa los datos a disco y solicita al lector la siguiente tarjeta. Ya que las tarjetas se almacenan en disco, cuando un programa durante su ejecución necesite los datos de una de ellas, se los solicitará al sistema operativo, que los leerá de disco y se los entregará al programa. Observemos que la diferencia entre esto y el mecanismo de buffers anterior es que ahora la CPU lee tarjetas con datos que no son procesados por el programa en ejecución, sino por programas que se ejecutarán posteriormente y que, como acabamos de decir, tomarán los datos del disco.

Veamos ahora el funcionamiento de la salida cuando esta se envía a una impresora. En este caso, la salida de un programa se guarda en disco, en vez de enviarse directamente a la impresora. Cuando el programa termina de enviar datos, toda su salida se encuentra en disco. A partir de ese momento, la CPU la va enviando poco a poco a la impresora: la CPU envía unos datos a la impresora y, mientras esta los procesa, la CPU continúa con la ejecución del programa en curso (evidentemente, la impresora debe tener algún tipo de almacenamiento temporal para guardar los datos que debe imprimir); después la CPU envía otro conjunto de datos y así sucesivamente.

Con los *spoolers*, conseguimos que la CPU y los dispositivos de E/S estén más ocupados. Además, la posibilidad de tener varios trabajos distintos en disco hace que sea posible cambiar el orden de procesamiento de los mismos. Sin embargo, este mecanismo tiene también sus limitaciones pues, por lo general, un único usuario no puede mantener todo el tiempo ocupados a la CPU y a los dispositivos de E/S. Es en la multiprogramación y en el tiempo compartido, que aparecen en la siguiente generación, donde esta

[7]Evidentemente, la CPU no es capaz de hacer nada por sí sola, solo ejecuta código. Por eso, cuando aquí hablamos de CPU debemos entender que es el sistema operativo, o programa similar, el que pasa a ejecutarse en la CPU y el que realmente realiza las acciones indicadas.

técnica mostrará todo su potencial.

1.3.3 Tercera generación (1965–1980): circuitos integrados y multiprogramación

La aparición de los circuitos integrados en esta generación hace que los ordenadores se vuelvan más pequeños y fiables, una tendencia que seguirá también en las siguientes generaciones.

Dos conceptos importantes de esta época son la multiprogramación y el tiempo compartido.

La *multiprogramación* consiste en tener varios programas en memoria. Cuando el programa que usa la CPU no pueda continuar su ejecución porque tiene que esperar (por ejemplo, a que termine una operación de E/S que él ha solicitado), se pasará a otro programa que esté listo para ejecutarse, es decir, que pueda utilizar la CPU. A su vez, cuando este tenga que esperar, se pasará la CPU a otro, y así sucesivamente. De esta manera, se consigue la superposición de la E/S de unos programas con el uso de la CPU de otros.

Puesto que la multiprogramación hace que, en un momento determinado, pueda haber en memoria principal varios programas que pueden ejecutarse, habrá que decidir a qué programa se le concederá el uso de la CPU. Es decir, surge la necesidad de hacer una *planificación de programas*. Pero, no solo eso, sino que también será importante evitar que los programas durante su ejecución interfieran unos con otros en el uso de los diferentes recursos: ficheros, impresora (donde no puede haber simultaneidad de uso), memoria, etc.

El *tiempo compartido* o *multitarea* es una variante de la multiprogramación. Consiste en realizar un cambio rápido entre tareas (o programas), lo que permite que cada usuario pueda interactuar con el programa que está ejecutando y tener la sensación de que él es el único usuario de la máquina.

Puesto que puede ocurrir que un programa de un usuario no quiera liberar la CPU durante un periodo grande de tiempo (lo que haría que se detuvieran los programas de otros usuarios), será necesario disponer de mecanismos para obligar a un programa a dejar la CPU cuando haya pasado cierto tiempo (generalmente, unas pocas milésimas de segundo).

Al igual que los sistemas con multiprogramación, los sistemas de tiempo compartido son sistemas complejos que deben gestionar varios programas que se ejecutan a la vez, lo que supone también arbitrar y planificar el uso de los diferentes recursos.

Recordemos que, en la primera generación, los ordenadores eran sistemas interactivos con un coste muy elevado, lo que obligó a usar diversas técnicas durante la segunda generación para aumentar el rendimiento del sistema (procesamiento por lotes, operación fuera de línea, etc.); estas técnicas, sin embargo, hicieron que el programador perdiera la interactividad con el mismo. Con los sistemas de tiempo compartido, los usuarios recuperan de nuevo la interactividad a un coste razonable.

En esta generación aparecieron sistemas operativos tan importantes como el OS/360 para el IBM 360 y MULTICS, el cual, aunque al final resultó ser un fracaso, sentó las bases que dieron paso a Unix, del que se derivan Linux y otros sistemas operativos similares.

1.3.4 Cuarta generación (1980-1995): ordenadores personales

Esta generación sigue avanzando en la mejora de los circuitos integrados, que se vuelven todavía más pequeños, dando lugar a la aparición de circuitos LSI (*Large Scale of*

Integration) y VLSI (*Very Large Scale of Integration*). Ahora, en un chip relativamente pequeño, es posible integrar millones de transistores y otros componentes electrónicos.

Gracias a la aparición de este tipo de circuitos, el precio de los ordenadores desciende de manera notable, haciéndolos cada vez más asequibles. Es en esta generación donde aparece el ordenador personal o PC (*Personal Computer*), gracias al cual se popularizaron los computadores y la informática.

También en esta generación aparecieron las estaciones de trabajo que, en realidad, son ordenadores personales muy potentes, habitualmente conectados entre sí por medio de una red cableada de tipo Ethernet o similar.

Los sistemas operativos que dominan los ordenadores personales y las estaciones de trabajo durante esta generación son MS-DOS y Unix, respectivamente. En el caso de los ordenadores personales, son habituales la presencia de sistemas monousuario y monotarea (como MS-DOS), aunque con la aparición de OS/2, Windows 95/98 y otros sistemas operativos, comienzan a popularizarse conceptos como la multitarea, la memoria virtual, etc., ya tradicionales en las estaciones de trabajo con Unix.

Dentro de esta generación, a mediados de la década de los 80, empiezan a proliferar las redes de ordenadores personales, lo que da lugar a la aparición de nuevos tipos de sistemas operativos, como los sistemas operativos de red y los sistemas operativos distribuidos. Asimismo, aparecen ordenadores con varias CPU y los sistemas operativos de tiempo real. Comentaremos estos tipos de sistemas operativos en la siguiente sección.

1.3.5 Quinta generación (1995–actualidad): Internet y dispositivos móviles

Si algo caracteriza a esta generación es la explosión del uso de Internet (y de sus servicios, como el *cloud computing*) y la proliferación de dispositivos móviles de todos tipo, especialmente los llamados *teléfonos inteligentes* o *smartphones*. El primer teléfono inteligente, que combinaba telefonía y computación, fue el Nokia N9000, lanzado a mitad de los 90, si bien estos dispositivos no se popularizaron hasta el lanzamiento del iPhone de Apple en 2007.

En lo que a sistemas operativos se refiere, durante esta generación aparecen Windows 2000 y una versión ligeramente mejorada de este sistema operativo llamada Windows XP. Mientras que Windows 2000 y sus sucesores (Windows Server 2003, Windows 2008, etc.) se destinaron a los servidores del mundo empresarial, Windows XP y sus sucesivas versiones (Windows Vista, Windows 7/8/10, etc.) se destinaron a los usuarios. Otros sistemas operativos que alcanzan gran popularidad en esta generación son Linux, Apple Mac OS X (llamado actualmente macOS), Android e iOS.

1.4 TIPOS DE SISTEMAS OPERATIVOS

En la actualidad, existen una gran variedad de sistemas de cómputo, desde grandes supercomputadores hasta ordenadores más pequeños que una moneda. Para cada uno de estos sistemas, existen uno o más sistemas operativos que se pueden usar, de la misma manera que también hay sistemas operativos que se pueden ejecutar en diferentes tipos de computadores. A continuación, vamos a comentar los principales tipos de sistemas operativos y su relación con algunos de los sistemas de cómputo existentes.

1.4.1 Sistemas operativos de propósito general

En primer lugar, tenemos los *sistemas operativos de propósito general*, es decir, sistemas operativos diseñados para realizar una gran variedad de tareas computacionales en entornos de diferente naturaleza. Estos sistemas se caracterizan por su flexibilidad y

su capacidad para adaptarse tanto al hardware sobre el que se ejecutan como a los trabajos que han de procesar. Dentro de esta categoría encontramos Unix (principalmente, Linux en alguna de sus diferentes distribuciones), Microsoft Windows y Apple macOS.

Aunque, en teoría, todos los sistemas operativos de propósito general mencionados se pueden usar en diferentes sistemas de cómputo, en la práctica, dependiendo del sistema, encontraremos unos u otros. Así, en los *supercomputadores*, el sistema operativo más usado es Unix. Los supercomputadores son sistemas formados por un gran número de elementos de todo tipo (procesadores, memoria, discos, etc.), conectados entre sí mediante una red, que son capaces de procesar inmensas cantidades de datos en poco tiempo.

Unix también es el sistema operativo más común en los *mainframes*. Estos sistemas son capaces de procesar a la vez muchos trabajos que requieren enormes cantidades de E/S (sobre todo, disco y red). Los *mainframes* suelen servir tres tipos de trabajos: por lotes o *batch* (trabajos que no interactúan con los usuarios), transacciones (bancarias, de reservas de avión, etc.) y multitarea (para los trabajos de los usuarios que están interactuando con el sistema).

Cuando pasamos a sistemas de cómputo menos potentes, la variedad de sistemas operativos es mayor. Así, en los *servidores* podemos encontrar tanto Unix, como Windows, como macOS. Los servidores son computadores que sirven a múltiples usuarios a la vez a través de una red para compartir algunos de sus recursos hardware (como impresoras), y software (aplicaciones, ficheros, bases de datos, páginas web, etc.).

También encontramos esos mismos tres sistemas operativos en los *ordenadores personales* que, en la actualidad, no son sino pequeños *multiprocesadores*, ya que su procesador cuenta con varios núcleos o *cores* (que equivalen a los distintos procesadores de un multiprocesador) que comparten la memoria principal. Dos de las características principales de estos sistemas son la multitarea y el estar diseñados para proporcionar un buen rendimiento a un único usuario que ejecuta decenas de programas.

Finalmente, también encontramos versiones especiales de Unix, Windows y macOS en sistemas de cómputo de dimensiones reducidas, como los *teléfonos inteligentes*, *tabletas* y diferentes *sistemas integrados*. Estos sistemas no son más que pequeños ordenadores con menos recursos hardware que un ordenador común, pero también con un consumo de energía mucho menor. Al ser ordenadores, los sistemas operativos que encontramos son, en gran medida, los mismos que los que tenemos para los ordenadores personales, aunque adaptados a estos dispositivos y a sus pantallas táctiles.

1.4.2 Sistemas operativos de red

Todos los sistemas operativos que acabamos de mencionar en el apartado anterior son también *sistemas operativos de red*, aunque, dado lo común de la situación, rara vez se denominan así. En los sistemas operativos de red, los usuarios son conscientes de la existencia de varios ordenadores conectados mediante una red. Cada máquina es fundamentalmente independiente de las demás, ya que ejecuta su propio sistema operativo local y tiene sus propios usuarios. Sin embargo, gracias a la red, puede interactuar con el resto de máquinas para compartir información, recursos, etc., siempre que se hayan concedido las autorizaciones oportunas entre las distintas máquinas que participan.

1.4.3 Sistemas operativos distribuidos

Otro tipo son los *sistemas operativos distribuidos*. Al igual que los anteriores, necesitan un conjunto de computadores conectados entre sí mediante una red. Sin embargo, a diferencia de ellos, son vistos por los usuarios como un sistema tradicional, con un

único computador y un único sistema operativo. Es decir, se ofrece a los usuarios una *imagen única del sistema*. Estos sistemas son capaces de manejar varios procesadores y en ellos el usuario no es consciente del lugar donde se ejecutan sus programas ni dónde se encuentran sus ficheros (a esto se le conoce como *transparencia de localización*). Existen varias razones para la existencia de sistemas distribuidos:

- Compartición de recursos: puedo utilizar una impresora o cualquier otro recurso conectado en otra máquina.
- Aceleración de cálculos: al disponer de varios procesadores, puedo hacer que un trabajo use varios de ellos a la vez y así terminar antes.
- Confiabilidad o tolerancia a fallos: si una máquina falla, el resto seguirá realizando todo el trabajo, etc.

Ejemplos de sistemas operativos distribuidos son Amoeba, Plan 9 e Inferno (derivado de Plan 9). El problema de todos ellos es que, o bien han dejado de desarrollarse (como Amoeba), o bien hace tiempo que no se actualizan (como pasa con Plan 9 e Inferno).

1.4.4 Sistemas operativos de tiempo real

Además de los sistemas operativos que acabamos de comentar en los apartados anteriores, existen otros con características especiales. Así, encontramos los *sistemas operativos de tiempo real*, como VxWorks y QNX. En ellos, el parámetro clave son las restricciones temporales. Los hay de dos tipos: rigurosos y no rigurosos. En los primeros, una acción se debe realizar necesariamente en cierto momento o intervalo. De no cumplirse esta restricción, se pueden producir situaciones críticas que pueden poner el sistema en peligro. En los segundos, es aceptable no cumplir de vez en cuando un plazo, siempre y cuando el comportamiento general del sistema se ajuste a unos ciertos parámetros (como, por ejemplo, un determinado porcentaje de fallos).

1.4.5 Sistemas operativos para tarjetas inteligentes

Otros sistemas especiales son los *sistemas operativos para tarjetas inteligentes* (las que tienen el tamaño de una tarjeta de crédito). Por las propias características de estas tarjetas, que tienen grandes limitaciones de potencia y memoria, estos sistemas operativos solo son capaces de realizar unas pocas funciones. Habitualmente, disponen de una máquina virtual de Java (JVM) que se encarga de ejecutar los *applets* que se cargan en la tarjeta[8]. En algunos casos, es posible cargar y ejecutar varios *applets* a la vez, por lo que son necesarios mecanismos de multiprogramación y planificación.

1.5 COMPONENTES Y SERVICIOS DE LOS SISTEMAS OPERATIVOS

Un sistema operativo proporciona un entorno dentro del cual se ejecutan los programas. Para construir este entorno, se hace necesario dividir lógicamente el sistema operativo en pequeños módulos y crear una interfaz bien definida (mediante llamadas al sistema) para que los programas se comuniquen con el sistema operativo. Internamente, los sistemas operativos varían mucho en su estructura, organizándose según diferentes

[8] Un *applet* es una pequeña porción de código que no se puede ejecutar por sí misma, sino que necesita de un entorno, proporcionado por otro programa, para su ejecución. Un *applet* de Java, por tanto, es una porción de código escrito en Java.

esquemas; incluso es posible que diversos sistemas operativos definan una interfaz idéntica a pesar de tener una estructura interna diferente (lo que ocurre con diversos sistemas Unix).

En esta sección vamos a describir brevemente cuáles son los principales componentes de un sistema operativo, qué servicios ofrecen esos componentes a los programas y usuarios, y a través de qué mecanismo se ofrecen los servicios. Observa, por tanto, que vamos a hablar de diversos conceptos (procesos, ficheros, etc.) a través de tres puntos de vista:

- Los componentes internos del sistema operativo que dan soporte a esos conceptos.
- Los usuarios y programas que los utilizan y gestionan.
- Las llamadas al sistema a través de las cuales el sistema operativo ofrece los servicios asociados.

1.5.1 Componentes del sistema

Solo es posible crear un sistema tan grande y complejo como un sistema operativo dividiéndolo en fragmentos más pequeños. Cada uno de estos debe ser una porción bien definida del sistema que tenga sus funciones, entradas y salidas cuidadosamente establecidas. En los siguientes apartados vamos a ver algunos de los componentes que constituyen un sistema operativo moderno. Todos los componentes están interrelacionados, por lo que es muy difícil hacerse una idea completa de cada uno de ellos sin conocer los demás.

Administración de procesos

De forma muy general, podemos decir que un *proceso* es un programa en ejecución, es decir, es una entidad activa, ya que, entre otras cosas, tiene un contador de programa que dice cuál es la siguiente instrucción a ejecutar. Un programa, en cambio, es una entidad pasiva, pues es el contenido estático (código y datos) de un fichero almacenado en disco.

Un proceso es la unidad de trabajo de un sistema, ya que este asigna recursos (memoria, CPU, espacio en disco, uso de red, etc.) a los procesos para que puedan realizar su tarea.

En un sistema habrá varios procesos en ejecución, algunos de los cuales serán procesos del sistema operativo (aquellos que ejecutan código del sistema) y los demás serán procesos de usuario (aquellos que ejecutan código del usuario). Potencialmente, todos estos procesos pueden ejecutarse de forma concurrente, multiplexando la CPU entre ellos (es decir, la CPU se concede a un proceso durante un tiempo para después pasar a dársela a otro proceso durante otro tiempo determinado y así sucesivamente).

El sistema operativo es responsable, entre otras, de las siguientes actividades relacionadas con la administración de procesos:

- Crear y eliminar procesos de usuario y de sistema.
- Repartir el uso de la CPU entre los distintos procesos.
- Suspender y reanudar la ejecución de procesos.
- Proporcionar mecanismos para la sincronización y comunicación entre procesos.

Los procesos los describiremos en detalle en el tema 2.

Administración de la memoria principal

La memoria principal (RAM) es parte central del funcionamiento de un sistema de computación moderno. En ella se encuentran las instrucciones que ejecuta la CPU y en ella están los datos que se leen o escriben durante la ejecución de las instrucciones. La memoria principal se utiliza también para otras cosas, como la E/S realizada mediante DMA, buffers, etc.

Para ejecutar un programa necesitamos que se encuentre total o parcialmente en la memoria principal. Además, como se vio en la multiprogramación, para mejorar la utilización de la CPU y la velocidad de respuesta del ordenador a los usuarios, se deben mantener varios programas en memoria a la vez.

Puesto que la memoria es compartida entre los procesos y el sistema operativo, es necesario realizar un reparto de la misma de forma adecuada. Existen diferentes esquemas para la administración de memoria. La selección de un cierto esquema de administración de memoria depende de varios factores, pero, sobre todo, del diseño del hardware del sistema, ya que cada esquema requiere su propio apoyo del hardware.

Como veremos en el tema 5, algunas de las actividades relacionadas con la administración de memoria, de las que el sistema operativo es responsable, son las siguientes:

- Llevar un control de qué zonas de memoria se están usando y qué procesos las usan.
- Decidir qué procesos se cargarán en memoria cuando haya suficiente espacio disponible.
- Asignar y recuperar el espacio de memoria según se requiera.

Administración del sistema de E/S

Uno de los objetivos del sistema operativo es ocultar a los usuarios las particularidades de los dispositivos de E/S, ofreciendo, en la medida de lo posible, una interfaz homogénea que facilite el uso de los mismos. En Unix, por ejemplo, dichas particularidades se ocultan al resto del sistema operativo por medio de un sistema de E/S que consiste en:

- Una caché en memoria principal construida mediante buffers para acelerar las operaciones de lectura y escritura de disco.
- Manejadores específicos para los dispositivos hardware.
- Una interfaz general (y uniforme) con los manejadores de dispositivo que facilite la implementación de los mismos.

Entre los dispositivos de E/S más importantes se encuentran los que sirven de almacenamiento secundario. Como la memoria principal es demasiado pequeña para almacenar permanentemente todos los datos y programas, el sistema de computación debe ofrecer un almacenamiento secundario que respalde a la memoria principal. Generalmente, dicho almacenamiento secundario está constituido por discos de diferentes tipos (discos duros o magnéticos, discos SSD o de estado sólido, pendrives, etc.). Muchos programas, incluyendo compiladores, ensambladores y editores, se almacenan en disco hasta que se cargan en la memoria, y luego utilizan el disco como fuente y destino de su actividad (eso sí, a través del concepto de fichero). Por tanto, la adecuada administración del almacenamiento en disco (registro del espacio libre, asignación de espacio para

nuevos ficheros y/o nuevos datos, planificación del disco, etc.) también es importante para el sistema de computación.

La gestión de la E/S y el manejo de los discos se verá en el tema 6.

Administración de ficheros

En estrecha relación con los dispositivos de almacenamiento secundario (discos magnéticos, de estado sólido, discos ópticos, memorias USB, etc.) se encuentra la administración de ficheros, que es uno de los componentes más visibles de un sistema operativo.

Para usar cómodamente un sistema de computación, el sistema operativo ofrece una perspectiva lógica uniforme del almacenamiento de información, abstrayéndonos así de las propiedades particulares de los dispositivos subyacentes. Para conseguir esto, define una unidad de almacenamiento lógico que es el *fichero*. Gracias a ello, es posible acceder (es decir, leer y escribir) cómodamente a la información existente en los dispositivos físicos a través de los ficheros definidos sobre ellos.

Puesto que el número de ficheros que hay en un dispositivo puede ser grande, estos se organizan en *directorios* para facilitar su uso.

También hay que tener en cuenta que, si hay varios usuarios que pueden acceder a los ficheros, habrá que controlar de alguna manera quién tiene acceso a los ficheros y qué operaciones puede realizar sobre ellos.

El sistema operativo es, por tanto, responsable de las siguientes actividades relacionadas con la administración de ficheros:

- Creación y eliminación de ficheros y directorios.
- Cambio de nombre de un fichero o directorio y modificación de los atributos asociados.
- Soporte de operaciones para manipular el contenido de ficheros y directorios (leer información, escribir, etc.).
- Correspondencia entre ficheros y almacenamiento secundario.

En el tema 4 veremos las técnicas de administración de ficheros.

Sistema de protección

Los distintos procesos de un sistema operativo deben ser protegidos unos de otros. Con este fin se proporcionan mecanismos para asegurar que los ficheros, segmentos de memoria, la CPU y otros recursos puedan ser usados únicamente por aquellos procesos que han recibido la correspondiente autorización del sistema operativo.

Al hablar de protección nos estamos refiriendo a un mecanismo que nos permite controlar el acceso de los procesos (y, por ende, de los usuarios que los usan) a los recursos definidos en un sistema de computación: ficheros, impresoras, unidades de disco, etc. Este mecanismo debe ofrecer un medio para especificar los controles que se impondrán junto con la manera de ponerlos en práctica.

El concepto de protección va más allá de los tres mecanismos hardware vistos en la sección 1.2 y será explicado en detalle en el tema 3.

1.5.2 Servicios del sistema operativo

Un sistema operativo ofrece diferentes servicios para proporcionar un entorno amigable donde desarrollar y ejecutar programas. Por supuesto, los servicios específicos que se ofrecen varían de un sistema operativo a otro, pero podemos identificar varias clases comunes:

- Ejecución de programas. El sistema debe ser capaz de cargar en memoria un programa y ejecutarlo. El programa debe poder terminar su ejecución, ya sea de forma normal o anormal (por ejemplo, por una excepción).
- Operaciones de E/S. Un programa en ejecución puede requerir E/S y esta quizás un fichero o dispositivo. Para ciertos dispositivos pueden necesitarse funciones especiales que difieren de una simple lectura o escritura (como rebobinar una unidad de cinta o borrar la pantalla en un monitor). Dado que un programa de usuario no puede ejecutar directamente operaciones de E/S, el sistema operativo debe ofrecer alguna forma de llevarlas a cabo.
- Manipulación del sistema de ficheros. El sistema de ficheros es de especial interés. Es obvio que los programas necesitan leer y escribir ficheros, así como crearlos y eliminarlos basándose en su nombre. Asimismo, necesitan poder manipular directorios para organizar los ficheros.
- Comunicaciones. Un proceso puede necesitar intercambiar información con otro. Para conseguir esto existen varias técnicas, como la memoria compartida (rango de direcciones de memoria dentro de un proceso donde lo que escribe el proceso puede ser leído por otro y viceversa), las tuberías (ficheros especiales donde la información escrita por un proceso puede ser leía por otro), etc.
- Detección de errores. Bien durante la ejecución de programas, bien al utilizar cualquiera de los servicios anteriores, etc., pueden producirse errores. Estos errores pueden surgir, por ejemplo, al leer de memoria principal (por un error de paridad[9], ...), durante una operación sobre un dispositivo de E/S (un error de paridad en la cinta, falta de papel en la impresora, un dispositivo que no responde, ...) o en la ejecución de un programa de usuario (desbordamiento aritmético, intento de acceso a una posición ilegal de memoria, ...). El sistema operativo debe estar preparado para detectar y tratar todos los posibles errores, y emprender la acción adecuada, llegado el caso, para asegurar un funcionamiento correcto y consistente.

Existe, además, otro conjunto de funciones del sistema operativo pensadas no tanto para ayudar a los usuarios, sino para asegurar un funcionamiento eficiente del sistema, como son:

- Asignación de recursos. Cuando varios procesos se ejecutan al mismo tiempo, se deben asignar recursos (CPU, memoria principal, almacenamiento en ficheros) a cada uno de ellos. Para hacer el reparto de recursos existen varios algoritmos que estudiaremos en los siguientes temas.

[9] La paridad es una técnica sencilla de detección de errores que se puede implementar tanto por hardware como por software. Su funcionamiento es simple: al leer la información almacenada en un cierto elemento, se calcula su paridad (básicamente, se ve si el número de bits a uno es par o no). Si la paridad coincide con la calculada y almacenada cuando se hizo la escritura, se supone que la información leída es correcta; si no, se produce un error, ya que posiblemente lo leído no sea válido.

- Contabilidad. Cuando deseamos llevar un control del uso de los recursos del ordenador por parte de los usuarios. Este control puede tener fines contables (para facturar a los usuarios) o servir simplemente para recopilar estadísticas de uso que pueden ayudar a los administradores del sistema a la hora de reconfigurar el mismo y así mejorar los servicios, planificar posibles ampliaciones futuras, etc.

- Protección. Cuando, en un sistema multiusuario, cada usuario desea controlar el uso de la información que le pertenece (quién puede acceder, de qué manera, etc.). También, cuando varios procesos independientes se ejecutan al mismo tiempo, no debería ser posible que un trabajo interfiriera con los demás, ni con el propio sistema operativo. La protección implica revisar la validez de todos los parámetros que se pasan en las llamadas al sistema y asegurar que todo acceso a los recursos del sistema esté controlado.

  También es importante la seguridad del sistema respecto a personas ajenas. Esta seguridad comienza con las contraseñas que los usuarios deben especificar para tener acceso a los recursos, y se extiende a las conexiones remotas que se pueden iniciar desde otros sistemas con el fin de proteger al sistema de intentos de acceso no permitidos. Todas estas conexiones deben registrarse a fin de detectar posibles intromisiones.

  Como vemos, para que un sistema esté bien protegido y seguro, hay que establecer controles en todas partes. En definitiva, una cadena es tan fuerte como su eslabón más débil.

1.5.3 Llamadas al sistema

Las llamadas al sistema definen la interfaz entre el sistema operativo y un programa en ejecución. Estas llamadas, generalmente, son instrucciones en ensamblador[10] que hacen que la ejecución pase de modo usuario a modo núcleo y se salte a una porción de código concreta del sistema operativo para iniciar el procesamiento de los servicios solicitados. El funcionamiento concreto se muestra en la figura 1.5 y consiste básicamente en tres pasos:

- En primer lugar (paso 1), se ejecuta la instrucción de llamada al sistema, produciendo que la máquina cambie de modo usuario a modo núcleo y se transfiera el control al sistema operativo. El sistema operativo examina entonces los parámetros de la llamada para determinar cuál de ellas se desea realizar.

- A continuación (paso 2), el sistema operativo analiza una tabla que contiene en la entrada k un apuntador (o puntero) al procedimiento que realiza la k-ésima llamada al sistema. Con ello se identifica el procedimiento de servicio al que se llama para efectuar la operación deseada.

- Por último (paso 3), la llamada al sistema termina y el control regresa al proceso de usuario, que continúa su ejecución por la instrucción inmediatamente posterior a la llamada al sistema.

Podemos apreciar que no existe una llamada al sistema para cada tipo de solicitud (sea esta solicitud de E/S o de cualquier otra cosa). En su lugar, el usuario indica, por ejemplo, por medio de uno de los registros del procesador[11], el número de servicio a realizar.

[10] Recordemos, por ejemplo, la instrucción `syscall` de MIPS.

[11] En MIPS se utiliza el registro `$v0` para especificar el número de llamada al sistema.

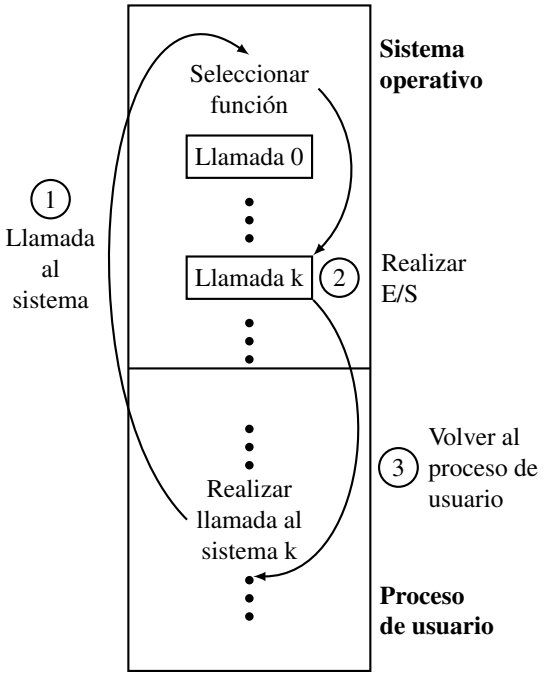


Figura 1.5: Uso de una llamada al sistema para efectuar una operación de E/S.

Mediante las llamadas al sistema, los programas obtienen los servicios que les proporciona el sistema operativo. Por tanto, y desde un punto de vista general, podemos decir que un sistema operativo no es más que un conjunto de llamadas al sistema.

Las llamadas al sistema pueden efectuarse de varias maneras, dependiendo del computador que se use. Frecuentemente, se requiere más información que la identidad de la llamada al sistema que se desea. El tipo y cantidad exacta de información varían en función del sistema operativo y la llamada concreta. Por ejemplo, para obtener datos de entrada, quizás tengamos que especificar el fichero o dispositivo que se usará como fuente, así como la dirección y la longitud del buffer de memoria donde se almacenará lo leído. Aunque no es lo habitual, en ocasiones, toda esta información puede estar implícita en la propia llamada al sistema.

El paso de parámetros al sistema operativo durante una llamada al sistema se puede realizar de tres formas generales:

- En registros. Es la más sencilla, aunque en algunos casos puede haber más parámetros que registros.
- En un bloque de memoria. Los parámetros se almacenan en un bloque en la memoria, y la dirección del bloque se pasa como parámetro en un registro (ver figura 1.6).
- En la pila. Los parámetros se introducen en la pila del programa que realiza la llamada al sistema, de donde los extrae el sistema operativo.

Las bibliotecas de los lenguajes de alto nivel suelen contener procedimientos que ocultan las llamadas al sistema, proporcionando así una interfaz mucho más sencilla que la proporcionada por las llamadas al sistema en lenguaje ensamblador. Por ejemplo, `printf` es una función de C que construye una cadena según el formato y los parámetros que recibe, efectúa las llamadas al sistema necesarias para mostrar la cadena por

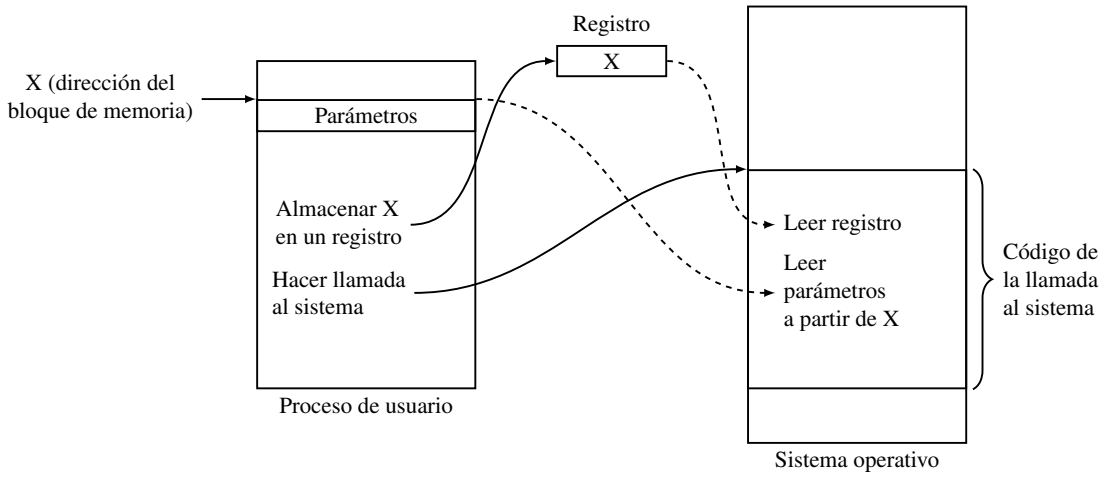


Figura 1.6: Paso de parámetros mediante un bloque de memoria.

pantalla, comprueba los posibles errores que se puedan producir y, finalmente, devuelve el control al programa de usuario. De esta forma, el compilador del lenguaje y sus bibliotecas de apoyo ocultan al usuario la mayor parte de los detalles de la interfaz con el sistema operativo.

Las llamadas al sistema se pueden clasificar en cinco grandes categorías, como se puede ver en la figura 1.7. En los siguientes temas veremos la teoría que nos permitirá comprender mejor dichas llamadas, por lo que no profundizaremos más aquí.

1.5.4 Programas del sistema

Otro aspecto de un sistema moderno son sus *programas del sistema*. Estos son programas que acompañan al sistema operativo para ofrecer un entorno más cómodo para el desarrollo y la ejecución de programas, y pueden dividirse en varias categorías:

- Manipulación de ficheros. Estos programas crean, eliminan, copian, renombran, imprimen, vuelcan, muestran y, en general, manipulan ficheros y directorios.

- Información de estado. Algunos programas simplemente solicitan al sistema información sobre la fecha, hora, espacio de memoria o disco disponible, número de usuarios, etc. A esta información se le da formato y se imprime en la terminal[12] u otro dispositivo, o se guarda en un fichero.

- Apoyo a lenguajes de programación. Habitualmente, con el sistema operativo se ofrecen compiladores, ensambladores e intérpretes para los lenguajes de programación habituales. Muchas veces estos programas vienen acompañados de depuradores.

- Comunicaciones. Estos programas proporcionan mecanismos para crear conexiones virtuales entre procesos, usuarios y diferentes sistemas de computación; permiten a los usuarios enviar mensajes a las pantallas de los demás, enviar mensajes de mayor tamaño en forma de correo electrónico o transferir ficheros de una máquina a otra; incluso permiten conectarse de forma remota a otros computadores.

[12] Un terminal, o una terminal, es un dispositivo lógico o físico, dotado de pantalla y teclado, mediante el cual se proporcionan datos a un ordenador o se obtiene información de él.

- Control de procesos:
  - Finalizar, abortar.
  - Cargar, ejecutar.
  - Crear proceso, terminar proceso.
  - Obtener/establecer atributos de proceso.
  - Esperar un tiempo, esperar un suceso.
  - Enviar señal a un proceso.
- Manipulación de ficheros:
  - Crear/eliminar fichero.
  - Abrir, cerrar.
  - Leer, escribir, reposicionar.
  - Obtener/establecer atributos de fichero.
- Manipulación de dispositivos:
  - Solicitar/liberar dispositivo.
  - Leer, escribir, reposicionar.
  - Obtener/establecer atributos de dispositivo.
- Mantenimiento de información:
  - Obtener/establecer hora o fecha.
  - Obtener/establecer datos del sistema.
  - Obtener/establecer atributos de proceso, fichero o dispositivo.
- Comunicaciones:
  - Crear/Eliminar conexión para la comunicación.
  - Enviar/Recibir mensajes.
  - Transferir información de estado.

Figura 1.7: Tipos de llamadas al sistema

- Programas de aplicación. La mayoría de los sistemas operativos se entregan con programas útiles para solucionar problemas o efectuar operaciones comunes. Estos programas incluyen editores y procesadores de texto, generadores de gráficos, sistemas de bases de datos, hojas de cálculo, paquetes de análisis estadístico, juegos, etc.

Como los programas del sistema no dejan de ser programas comunes, desde el punto del sistema operativo *no existe diferencia* entre un programa de usuario (un programa instalado o desarrollado por nosotros mismos) y un programa del sistema.

Quizás, el programa de sistema más importante para un sistema operativo sea el *intérprete de mandatos u órdenes* (*shell* en inglés), cuya función principal es obtener la siguiente orden especificada por el usuario y ejecutarla. La forma de interactuar con un intérprete de órdenes suele ser a través de un programa terminal o consola en modo gráfico (como Gnome Terminal y Konsole en Linux, o Consola de Windows y Terminal de Windows en el sistema operativo Windows), aunque también lo podemos hacer mediante una terminal de texto.

Mientras que las llamadas al sistema definen la interfaz con el programador, la perspectiva del sistema operativo que tienen la mayoría de los usuarios es la definida por

los programas del sistema, de modo que esta perspectiva puede estar bastante alejada del sistema real. Por consiguiente, el diseño de una interfaz útil y amigable con el usuario no programador depende de los programas del sistema que acompañan al sistema operativo, y no es una función directa de este.

1.6 ESTRUCTURA DE LOS SISTEMAS OPERATIVOS

Como hemos visto, un sistema operativo es un programa (muy especial, eso sí) formado por distintos componentes que ofrecen servicios a los programas y usuarios que hacen uso de él. Es evidente que este programa y sus componentes deben estar organizados de alguna manera y tener algún tipo de estructura. En esta sección vamos a mostrar brevemente algunas de esas posibles estructuras internas de los sistemas operativos, examinando tres que ya han sido probadas en la práctica. Estas estructuras no son las únicas, pero sí son sencillas de entender.

1.6.1 Sistemas monolíticos

Este tipo de organización es, con mucho, la más común. Se caracteriza por:

- Estar formada por una colección de procedimientos que se llaman unos a otros.
- Tener cada procedimiento definida una interfaz muy clara en cuanto a lo que hace el procedimiento, los parámetros que recibe y los resultados que produce.
- No tener una estructura bien definida de los procedimientos existentes.
- No existir ocultación, ya que todo procedimiento es visible a los demás.

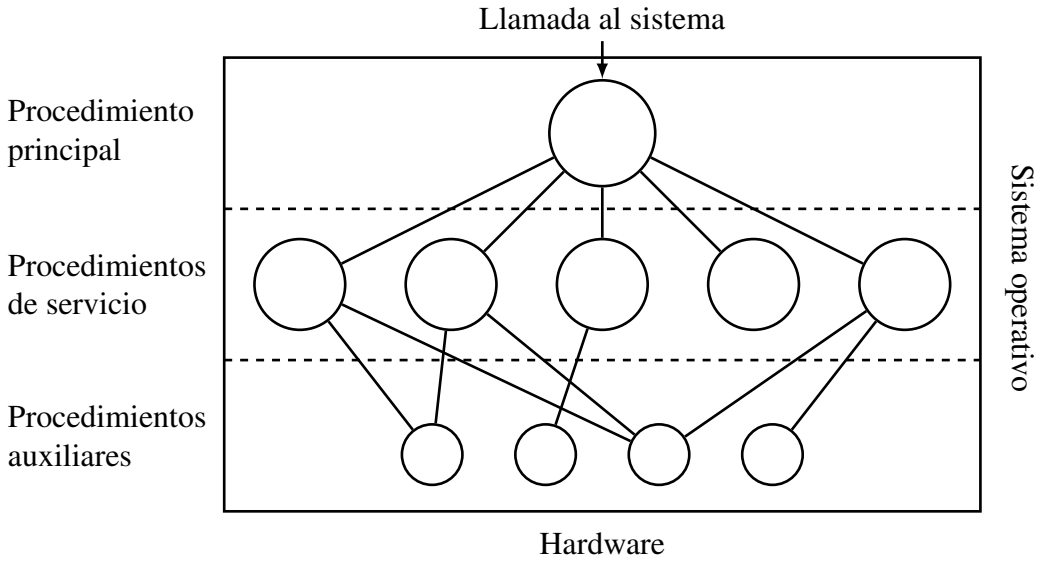


Figura 1.8: Un modelo de estructura simple para un sistema monolítico.

Aunque hemos dicho que no hay una estructura bien definida, es evidente que algo de estructura debe haber. Teniendo en cuenta que los programas de usuario solicitan servicios al sistema operativo mediante llamadas al sistema (como hemos visto en la sección 1.5.3), podemos pensar que un sistema operativo monolítico tiene una estructura básica como la mostrada en la figura 1.8, donde se puede ver:

- Un procedimiento principal, que recibe las peticiones de llamadas al sistema y que, tras analizarlas, llama a los procedimientos de servicio correspondientes.
- Un conjunto de procedimientos de servicio, que llevan a cabo las llamadas al sistema.
- Un conjunto de procedimientos auxiliares, que ayudan a los procedimientos de servicio a realizar su trabajo.

Ejemplos de sistemas operativos actuales con esta estructura son muchos sistemas Unix (incluyendo Linux) y Windows[13].

1.6.2 Sistemas con capas

Los avances en el hardware han proporcionado nuevas facilidades a los implementadores de sistemas operativos que les han permitido realizar un diseño más modular de los mismos. Un ejemplo de diseño modular es el basado en capas.

En el diseño modular por capas (ver figura 1.9) el sistema operativo se divide en capas (niveles), cada una construida sobre la anterior. La capa más baja (capa 0) corresponde al hardware y la capa más alta (capa *N*) es la interfaz con el usuario.

La principal ventaja de esta estructura es la modularidad:

- Cada capa solo utiliza las funciones y servicios de la capa inmediatamente inferior, los cuales puede exportar directamente u ocultar a las capas superiores, o utilizar para implementar nuevas operaciones que también se ofrecerán a las capas superiores (ver figura 1.9).
- Se simplifican la depuración y verificación: una vez se ha implementado, verificado y depurado una capa, se asciende a la siguiente y se repite el proceso.
- Cada capa oculta los detalles de bajo nivel a las capas superiores. Esta ocultación de información permite modificar una capa en cualquier momento sin tener que hacer cambios en otras capas, siempre que no se altere la interfaz de la capa.

La mayor dificultad de este enfoque es el propio diseño de las capas (número de capas, funciones de cada una, interfaz exportada por cada capa, etc.), ya que cada capa solo puede utilizar los servicios de las capas inferiores. Esto hace que surjan problemas de dependencias entre capas y que, a veces, sea difícil saber dónde colocar una determinada función, por lo que en la actualidad se opta por tener pocas capas, pero con más funcionalidad en cada una.

Un ejemplo de sistema operativo con esta estructura fue THE (1968), que se definió en 6 capas (ver figura 1.10). Otro ejemplo fue el sistema operativo VENUS, donde las capas inferiores se implementaron directamente en hardware, haciendo que su funcionamiento fuera más rápido.

[13]También podemos afirmar que tanto Linux como Windows tienen una *estructura modular*, ya que es posible desarrollar de forma independiente porciones de código del sistema operativo (*módulos*) que se podrán cargar y descargar dinámicamente en tiempo de ejecución. A pesar de ello, una vez cargados, estos módulos forman parte del código del sistema operativo en memoria a todos los efectos y, como el resto del código del mismo, se ejecutan en modo núcleo sin ningún tipo de protección ni ocultación. Por tanto, a pesar de la existencia de estos módulos, el funcionamiento del núcleo del sistema operativo sigue siendo el de uno monolítico.

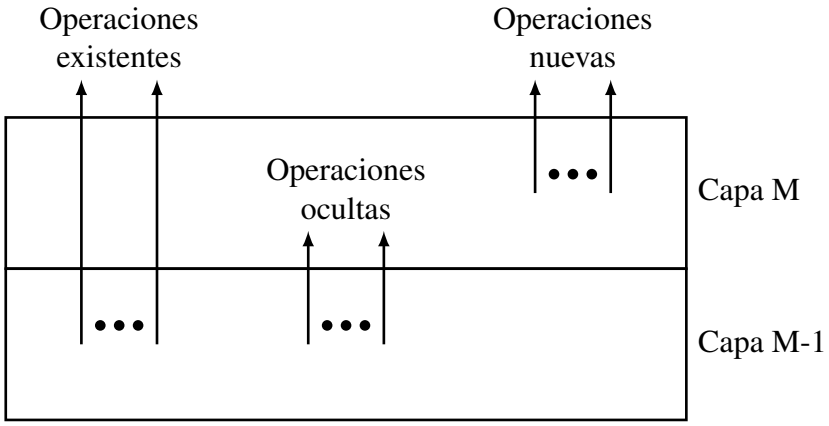


Figura 1.9: Implementación de la capa M en función de la capa M-1.

Capa	Descripción
5	Programas de usuario
4	Administración de la E/S
3	Consola del operador
2	Administración de memoria
1	Planificación de la CPU
0	Hardware

Figura 1.10: Estructura de capas de THE.

1.6.3 Modelo cliente-servidor

Una tendencia de los sistemas operativos modernos es la de explotar la idea de mover parte del sistema operativo a capas superiores y mantener un núcleo mínimo, llamado micronúcleo o *microkernel*. Siguiendo este modelo, lo usual es implantar la mayoría de las funciones del sistema operativo en los procesos de usuario, por lo que se distinguen dos tipos de procesos:

- Clientes: solicitan servicios.
- Servidores: realizan los trabajos solicitados por los procesos clientes y les devuelven las respuestas.

La principal función del micronúcleo es controlar las comunicaciones cliente-servidor, como se aprecia en la figura 1.11. Generalmente, los procesos clientes son los de usuario, pero un proceso servidor también puede ser cliente si solicita servicios a otro proceso servidor.

Los servidores se ejecutan en modo usuario, por lo que no hay acceso directo al hardware. Esto tiene grandes ventajas. Por ejemplo, un fallo en el sistema de ficheros no afecta a todo el sistema. Incluso es posible reiniciar el proceso servidor y hacer que el sistema se recupere de la caída del sistema de ficheros.

Otra ventaja del modelo cliente-servidor es su capacidad de adaptación a los sistemas distribuidos. Un cliente se comunica con el servidor por medio de mensajes, sin importar si ambos están en la misma o en diferentes máquinas (ver figura 1.12); en lo que respecta al cliente, ocurre lo mismo en ambos casos: se envía una solicitud y se recibe una respuesta.

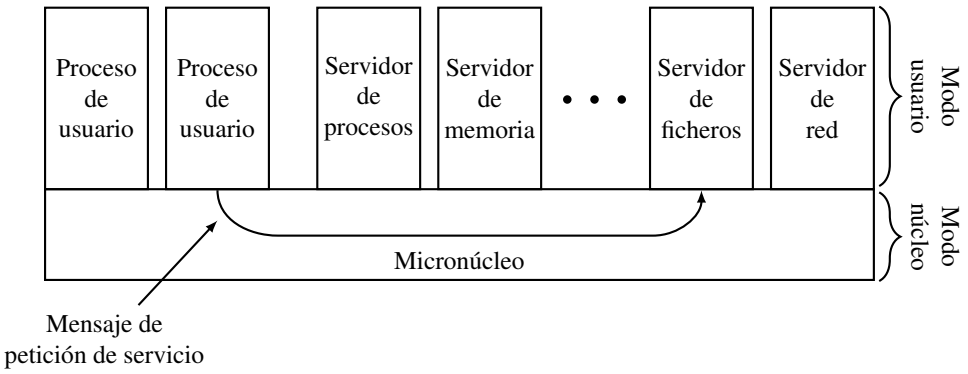


Figura 1.11: Modelo cliente-servidor. Los procesos clientes obtienen servicios al enviar mensajes a los procesos servidores.

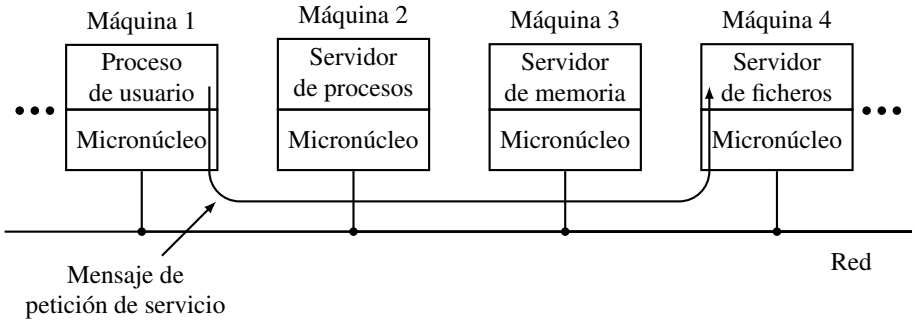


Figura 1.12: El modelo cliente-servidor en un sistema distribuido.

Ya que los servidores se ejecutan en modo usuario, hay funciones que no pueden realizar directamente ellos mismos, como, por ejemplo, cargar datos en los registros de los dispositivos de E/S (esta suele ser una operación privilegiada que solo se puede hacer en modo núcleo o supervisor). Para solucionar este problema, existen dos opciones principales:

- Incluir el servidor en el núcleo. Esto hace que, al menos para el servidor incluido, desaparezca la ventaja de protección que este modelo ofrece frente a los fallos en el propio servidor.

- Existencia de mensajes especiales enviados por los servidores que captura el micronúcleo para procesarlos él mismo (por ejemplo, cargar valores en los registros de la controladora de disco[14]). En este caso, el micronúcleo no solo realizará las operaciones privilegiadas en nombre de los servidores, sino que también verificará si el servidor que le envía una solicitud tiene permiso para hacerlo.

De las dos soluciones, es evidente que es preferible la segunda, pues se mantienen todas las ventajas de ejecutar los servidores en modo usuario y, además, se permite que

[14]Una *controladora*, o *tarjeta controladora*, es un componente electrónico que hace de puente entre un dispositivo hardware y el resto del sistema, es decir, una controladora conecta a un dispositivo hardware al sistema. Así, la controladora de un disco nos permite acceder a dicho disco a través de las órdenes que enviamos a la controladora. Estas ordenes habitualmente serán enviadas por el manejador de disco correspondiente. Es importante no confundir controladora con controlador, que es uno de los posibles nombres que puede recibir un manejador de dispositivo. Para evitar confusiones, nosotros siempre hablaremos de manejadores y controladoras (o tarjetas controladoras).

estos puedan, indirectamente con ayuda del micronúcleo, realizar aquellas operaciones privilegiadas que necesiten.

Podemos encontrar este modelo cliente-servidor con micronúcleo en sistemas operativos actuales como QNX, Minix3 o macOS[15].

[15]El núcleo de macOS tiene su origen en la versión 2.5 de Mach, un sistema operativo micronúcleo desarrollado en la década de los 80 en la Carnegie Mellon University.