1 Introducción

Se define la función del sistema operativo como gestor de recursos en contraste con su papel de interfaz con las aplicaciones. Desde este punto de vista cobra relevancia el rendimiento, como medida de la calidad del diseño, la implementación y la configuración de un sistema operativo. Un aspecto fundamental del diseño de un sistema operativo es su estructura. Las propuestas avanzadas tienen que competir con la inercia y la experiencia acumulada con las estructuras clásicas. Finalmente, es preciso considerar los condicionantes que el entorno comercial impone sobre la difusión y el éxito de un determinado sistema operativo.

Contenido

1.1 El sistema operativo como gestor de recursos

Informalmente el sistema operativo es el conjunto de software que hace funcionar un computador o un dispositivo análogo, como un teléfono móvil o un robot[1]. Más formalmente vamos a considerar una doble definición. Por una parte, un sistema operativo proporciona la funcionalidad definida para la **interfaz de llamadas al sistema**, que presenta al computador como una **máquina virtual**. Por otra parte, el diseño, implementación y configuración de un sistema operativo, materia objeto de estos apuntes, está enfocado al objetivo de la **gestión eficiente de los recursos** del computador.

La interfaz de llamadas al sistema determina la forma en que las aplicaciones acceden a los servicios del sistema. Se presenta como una biblioteca de funciones[2], a menudo encapsulada por las primitivas de los lenguajes de programación. Una aplicación particular sobre ella es la **interfaz de usuario**, bien en forma de intérprete de comandos o, modernamente, de interfaz gráfica de usuario (GUI), que en buena medida determina la *personalidad* de sistema operativo de cara al usuario no especializado. Finalmente, hay que destacar la **interfaz del administrador**, construida en parte sobre llamadas al sistema de uso restringido, mediante la que el administrador del sistema configura los parámetros del sistema de cara a afinar el rendimiento y gestiona usuarios y derechos de acceso.

En lo que respecta al papel del sistema operativo como gestor de los recursos del computador, introduciremos primero una idea general del concepto de recurso. Entenderemos por recurso cualquier elemento del sistema susceptible de ser utilizado por las aplicaciones: procesadores, memoria, dispositivos, servicios del sistema… A grandes rasgos, y dependiendo de la naturaleza de los recursos, puede decirse que el sistema tiene que gestionar *eficientemente*[3] el uso del tiempo y del espacio. La **gestión temporal** afecta a los procesos (por ejemplo, planificación expulsora o no), a los dispositivos (por ejemplo, gestión de peticiones en el acceso al disco), y a la memoria (minimización de la probabilidad de fallo de página y de su tiempo de tratamiento). La **gestión espacial** se refiere fundamentalmente a dispositivos de almacenamiento, tanto volátil (gestión de memoria) como permanente (gestión del disco y del sistema de ficheros), pero también a los procesadores (asignación de procesadores a procesos).

---

[1] En adelante utilizaremos genéricamente el término computador.

[2] Suele conocerse como API (*Application Programming Interface*).

[3] Queda pendiente una definición precisa de *eficiencia*, que proporcionaremos más adelante.

Clásicamente, desde el punto de vista de los recursos que hay que gestionar, los sistemas operativos se estudian considerando los siguientes elementos: procesos, procesador, dispositivos, memoria y ficheros. Esta estructura es la que se sigue en estos apuntes. En cada tema se introducirán las **políticas** y **mecanismos** que se utilizan en los sistemas operativos. Entenderemos por políticas las reglas que determinan el comportamiento de un elemento del sistema, mientras que los mecanismos se refieren a las técnicas que permiten implementar las políticas. Por ejemplo, una política de sustitución de páginas FIFO en un sistema de memoria virtual puede implementarse bien mediante un mecanismo de listas de marcos, bien etiquetando cada marco con un registro del tick de carga. La elección de uno u otro mecanismo se basa en relaciones **rendimiento/coste** y depende en gran medida de la arquitectura soporte y del estado tecnológico.

Estudiaremos no sólo las políticas que implementan los sistemas actuales, sino también las que, aun estando actualmente en desuso, han tenido relevancia histórica. Esto se justifica por el hecho de que los parámetros tecnológicos se modifican con el tiempo, y técnicas cuya utilización no se justifica hoy en día podrían ofrecer buenos resultados en un estado tecnológico diferente.

1.2 Evaluación del rendimiento

El **rendimiento**[4] de un sistema operativo se refiere a la calidad de éste como gestor de recursos. El rendimiento presenta distintas facetas y para medirlo se utilizan **parámetros** como la **eficiencia** (aprovechamiento o tasa de utilización del recurso). Así, en gestión espacial es importante maximizar la capacidad de direccionamiento y minimizar el espacio perdido. En gestión temporal se usan también otros parámetros de rendimiento, como la latencia y el tiempo de finalización. El reparto equitativo de los recursos entre los procesos es a menudo un criterio a tener en cuenta.

La adopción de una determinada política o mecanismo con el objetivo de optimizar un determinado parámetro del rendimiento (por ejemplo, tiempo compartido para mejorar el tiempo de respuesta y el reparto equitativo del procesador), conlleva a veces el empeoramiento de otros parámetros (en nuestro ejemplo, la eficiencia). Esto es debido a que a veces los parámetros expresan características contrapuestas, pero sobre todo a que la implementación de una política, en general, añade un coste adicional (en tiempo o espacio), que se denomina **sobrecarga** (*overhead*). Ello conduce a la necesidad de establecer **compromisos** en la decisión de adoptar unas soluciones u otras en el diseño o configuración de un sistema operativo. Además, dentro de estos compromisos hay que considerar también el coste económico. Por ejemplo,

[4] En inglés *performance*. También se traduce como **prestaciones**. En algunas traducciones puede encontrase como *desempeño*.

instalar una mayor cantidad de memoria en un sistema con memoria virtual mejora los tiempos de ejecución de los programas a costa de un mayor desembolso económico inicial.

Los parámetros del rendimiento aplicados a un determinado sistema se suelen expresar, mejor que como valores **absolutos**, como una medida de la mejora o empeoramiento **relativos** con respecto a un sistema de referencia. Así, para estudiar la introducción de un mecanismo B en sustitución de uno A ya existente, hay que ponderar las mejoras y empeoramientos relativos de B con respecto a A para cada parámetro de rendimiento.

Para la obtención de valores de los parámetros del rendimiento de un sistema operativo habitualmente se utilizan métodos empíricos (utilizando trazas, si el sistema está disponible, o simulaciones, en caso contrario). También en algunos casos es posible un enfoque analítico, por ejemplo aplicando modelos de colas, aunque aquí no vamos a profundizar en este aspecto. En cualquier caso, la medida de un parámetro de rendimiento arrojará como resultado una distribución estadística, de la que la media será habitualmente la variable más significativa. Es importante también la varianza, pues es un indicador del grado de predecibilidad del comportamiento del sistema. Un objetivo será mantener varianzas bajas en los parámetros de rendimiento para asegurar un comportamiento acotado en el peor caso.

A continuación comentaremos algunas de las ideas fundamentales en la gestión del tiempo y del espacio, y los parámetros de rendimiento asociados. Vamos a definir aquí los parámetros de una manera general. Cuando abordemos el estudio de cada parte del sistema operativo, los particularizaremos para el caso concreto.

1.2.1 Gestión del tiempo

En general, describiremos el sistema operativo como un modelo de colas donde un proceso realiza una petición a un recurso, espera a ser atendido, es servido y nuevamente solicita el uso de otro recurso[5]. Algunos de los parámetros temporales de la evaluación del rendimiento están extraídos de la teoría de colas.

Una característica fundamental de la cola asociada a un recurso es la disciplina de acceso a la cola, que determina en gran parte la política de gestión del recurso. Los criterios para elegir una disciplina buscan equilibrar los beneficios que aporta en el uso del recurso con el coste de implementarla, casi siempre medidos en tiempos. La disciplina de acceso trivial, FIFO, es sencilla de implementar pero no suele optimizar el uso del recurso. Alternativamente se puede gestionar la cola atendiendo a las

[5] Ampliaremos esta descripción en capítulos sucesivos.

características específicas del recurso y al comportamiento de los programas en su acceso, lo que lleva a una gran variedad de políticas, que trataremos en su momento. Es muy habitual utilizar **prioridades** para determinar el acceso a la cola, que se pueden establecer de acuerdo a criterios diversos.

1.2.1.1 Parámetros de rendimiento

Definiremos una serie de parámetros comunes para medir el rendimiento en lo que afecta al tiempo, que podrán aplicarse a diferentes aspectos del sistema operativo, como por ejemplo, la gestión de procesos y su planificación, la gestión de memoria, el acceso a discos... Aunque en su momento habrá que particularizar en la aplicación de estos parámetros para la medida del rendimiento en recursos concretos del sistema, y a veces será necesario definir parámetros adicionales, describiremos aquí los que son de aplicación general:

Eficiencia temporal

La eficiencia temporal, Ef_t, expresa la tasa de uso efectivo del recurso en el tiempo. Una eficiencia alta indica un buen aprovechamiento del recurso y es indicativa de un buen rendimiento. Por uso efectivo se entiende que se está realizando trabajo útil. Denominaremos **tiempo de trabajo útil**, $t_{útil}$, al tiempo de uso efectivo. El tiempo que el recurso se está utilizando para labores de gestión (por ejemplo, ejecución del algoritmo que implementa una determinada política), se considera **tiempo perdido** (véase el Ejercicio 1). El tiempo de trabajo útil más el tiempo perdido nos da el tiempo total de referencia, T.

$$Ef_t = \frac{t_{útil}}{T}$$

Productividad (***throughput***) o **tasa de servicio**

Es una forma de medir la *velocidad* con la que trabaja un recurso. Se expresa como la cantidad de servicios por unidad de tiempo que el recurso es capaz de atender.

Latencia o **tiempo de respuesta**

Tiempo que hay que esperar para usar el recurso (intervalo entre el instante en que se solicita y el instante en que comienza a atender la petición).

Tiempo de servicio

Tiempo que se requiere para tratar una petición, o tiempo que un proceso está usando el recurso.

1.2.2 Gestión del espacio

La gestión del espacio es un aspecto fundamental en sistemas de almacenamiento, aunque también puede aplicarse en cierta medida a la gestión de procesadores en sistemas multiprocesador.

Un sistema de almacenamiento se caracteriza por un espacio lineal de direcciones físicas, con el que se establece una correspondencia desde un espacio lógico, que puede ser lineal o estructurado[6]. Las direcciones se expresan de acuerdo a una **unidad de direccionamiento**, por ejemplo el byte. Una de las tareas de la gestión del almacenamiento es la traducción de direcciones del espacio lógico al físico, cuyo rendimiento se expresa por parámetros temporales (tiempo que se tarda en proporcionar la traducción) y requiere a veces el uso de hardware de traducción específico (por ejemplo, el TLB en sistemas de memoria paginados).

1.2.2.1 Particionado

En lo que respecta estrictamente a la gestión del espacio por el sistema operativo, un aspecto fundamental es cómo se define la **unidad de gestión** del espacio, es decir, la cantidad mínima de espacio que el sistema es capaz de manejar[7], en particular para asignar o liberar[8]. Una opción es que el sistema maneje trozos o **particiones** del espacio de direcciones de cualquier longitud, expresada en unidades de direccionamiento (**particionado variable**). La alternativa es definir un **particionado fijo**, con unidades de tamaños predeterminados (uniformes o con varios tamaños). Mientras que con el particionado variable se busca ajustar la cantidad de espacio asignado al tamaño del objeto a almacenar, con el fijo se asigna una partición predeterminada de tamaño igual o mayor al del objeto. Ambos enfoques plantean una problemática muy diferente, como veremos.

Para conseguir una mayor flexibilidad e independencia del tamaño del objeto, lo habitual es definir particiones de pequeño tamaño y asignar al objeto un conjunto de ellas (contiguas o no). De esta forma el problema se reformula como el almacenamiento de los *trozos* del objeto en las particiones definidas. Si definimos las particiones de un tamaño fijo uniforme de p bytes, a un objeto de longitud e bytes se le asignarán en tal sistema $\lceil e/p \rceil$ particiones.

---

[6] Por ejemplo, un sistema paginado posee un espacio lineal de direcciones lógicas, que se traducen al espacio lineal de direcciones físicas de la memoria principal. El sistema de ficheros es un espacio lógico estructurado que se almacena en un disco, que se ve como un espacio lineal de bloques.

[7] Por ejemplo, una página en un sistema paginado.

[8] También la llamaremos **unidad de asignación**.

1.2.2.2 Parámetros de rendimiento

Fundamentalmente, utilizaremos los siguientes parámetros para medir el rendimiento espacial:

Capacidad de direccionamiento

Se refiere a la cantidad de almacenamiento que se puede representar y depende de dos factores: el tamaño de la unidad de direccionamiento y el número de direcciones que se pueden especificar. Este último parámetro depende fundamentalmente de la longitud de los campos usados para representar las direcciones, aunque existen otros factores limitantes que no dependen del sistema operativo, como las características del hardware (por ejemplo, anchura de los buses) y de la propia instalación (por ejemplo, memoria instalada). Si se utiliza una unidad de direccionamiento de d bytes y se destinan b bits para representar una dirección, obtendremos un espacio con una capacidad de direccionamiento (en bytes) de:

$$E = d2^b$$

Eficiencia espacial

La eficiencia espacial, Ef_e, se refiere a la tasa de ocupación efectiva del recurso de almacenamiento, es decir, la porción del espacio realmente aprovechada (**espacio ocupado**, Q), en relación al espacio disponible, M.

$$Ef_e = \frac{Q}{M}$$

La pérdida de eficiencia espacial (espacio desaprovechado) se ve de diferente manera según el espacio se gestione con particiones de tamaño fijo o variable, respectivamente:

Fragmentación interna

Cuando el almacenamiento se gestiona mediante particionado fijo, para almacenar un objeto es necesario asignar una partición cuyo tamaño sea igual o mayor que el del objeto. La fragmentación interna expresa el espacio desaprovechado dentro de las particiones. En general:

$$FR_i = \frac{A - Q}{A}$$

siendo A el espacio asignado total. Para el caso particular de un sistema de particiones de tamaño fijo p bytes, para almacenar un objeto P de longitud e

bytes, siendo $e >> p$, en media se perderá la mitad de la última partición asignada, es decir, $p/2$ bytes, siendo por tanto la fragmentación interna media del objeto P:

$$\overline{FR_i}(P) = \frac{p/2}{\lceil e/p \rceil p} = \frac{1/2}{\lceil e/p \rceil}$$

Nótese que esta expresión ofrece una estimación estadística para el caso particular (y bastante común) de objetos con un tamaño significativamente mayor que el de la partición. Para calcular con precisión la fragmentación interna media en un sistema completo es necesario conocer la distribución estadística de los tamaños de los objetos.

Fragmentación externa

Cuando el espacio asignado se adapta al objeto a almacenar, la liberación de una partición dejará un hueco que sólo podrá ser ocupado por otro objeto de tamaño igual o menor. Con el tiempo, la gestión con particiones de tamaño variable provocará un número creciente de huecos de tamaño progresivamente menor, y será cada vez menos probable poder aprovecharlos para almacenar nuevos objetos (se dice que el hay **degradación** en el uso del espacio). En ese momento, no es posible almacenar un objeto de tamaño e aunque el espacio total libre sea mayor ($M - A > e$), y la fragmentación externa se expresa como:

$$FR_e = \frac{M - A}{M}$$

La degradación del espacio en los sistemas de particiones de tamaño variable conduce a la necesidad de **compactar** (reubicar los objetos en posiciones contiguas para dejar un único hueco). La compactación implica pérdida de tiempo por la copia masiva de información, por lo que estos sistemas, aunque espacialmente más eficientes, en principio, que los de particiones fijas, presentan menor eficiencia temporal.

Aparte de la fragmentación introducida por el particionado fijo o variable, hay que computar también como pérdida de eficiencia la información necesaria para la propia gestión del espacio, en concreto para la **representación del espacio libre** (y, complementariamente, del ocupado), como veremos en el apartado siguiente.

1.2.2.3 Gestión del espacio libre

En la gestión del espacio libre, hay que considerar dos vertientes: cómo se representa y qué política de asignación se sigue.

Representación del espacio

Hay dos mecanismos alternativos de representación del espacio libre:

(a) **Mapa de bits**. Cada unidad de asignación de espacio se representa mediante un bit, que indica si está ocupada o libre. Para particionado fijo, habrá un bit por partición. Para particionado variable se necesita un bit por unidad de asignación, lo que conduce en la práctica a considerar unidades de asignación grandes para mantener el tamaño del mapa razonablemente pequeño[9].

(b) **Listas de huecos**. Los huecos se representan por elementos de una lista encadenada (convenientemente ordenada por tamaño), especificado para cada hueco su dirección y su longitud. Resulta adecuado para particionado variable.

Políticas de asignación

En particionado variable (y también en particionado fijo con particiones de distintos tamaños) es preciso buscar un hueco adecuado para el objeto que se va a almacenar. El objetivo es optimizar la eficiencia espacial y retardar la degradación. Son posibles diferentes políticas de asignación:

(a) ***First-fit***. Se elige el primer hueco donde quepa el objeto. Si el hueco se busca a partir de la asignación anterior, el algoritmo se denomina ***next-fit***.

(b) ***Best-fit***. Se elige el menor hueco donde quepa el objeto.

(c) ***Worst-fit***. Se elige el mayor hueco.

Las medidas experimentales del retardo de la degradación de la memoria no ofrecen diferencias significativas. First-fit y next-fit son menos costosas en tiempo (no requieren recorrer toda la lista o mapa de bits).

1.3 Estructura de un sistema operativo

Desde un punto de vista de máquina virtual puede establecerse una clasificación de los sistemas operativos basándose en la funcionalidad que éstos ofrecen. Se habla entonces de sistemas operativos por lotes o interactivos, mono o multi-usuario, mono o multi-programados, mono o multi-puesto, etc. Desde el punto de vista del diseño, hay que fijarse sobre todo en la **estructura** del sistema operativo.

[9] Obsérvese que si se tomara el byte como unidad de asignación, el mapa de bits ocuparía la octava parte del espacio de almacenamiento físico.

En lo referente a su estructura, un sistema operativo es meramente un programa en el sentido de que está integrado por estructuras de datos y por las funciones que las utilizan. La particularidad del sistema operativo es que ocupa un **espacio de direcciones protegido**, proporcionando a las aplicaciones un medio de acceso homogéneo a los servicios que ofrece, mediante un mecanismo de *trap* que permite el cambio a **modo de ejecución privilegiado**[10] para la ejecución de las llamadas al sistema.

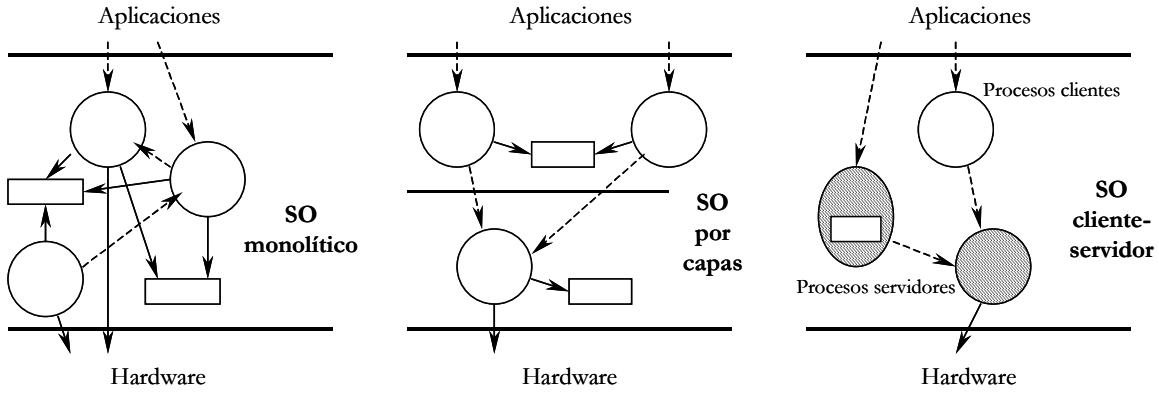

Figura 1.1. Estructuras de sistemas operativos. Los círculos representan procesos y funciones, los rectángulos, estructuras de datos.

Desde el punto de vista de su estructura, pueden distinguirse tres tipos de sistemas operativos, representados en la Figura 1.1:

Monolíticos

No existe una estructura definida. Cualquier función del sistema operativo puede acceder a cualquier estructura de datos y puede llamar, en principio, a cualquier otra función. Diseñar un sistema operativo multiprogramado monolítico es complejo y resulta prácticamente imposible controlar todas las situaciones de error y de interbloqueo. Cualquier modificación afecta a una gran parte del sistema. Pese a todo, a lo largo de la historia de los sistemas operativos los monolíticos son mayoría. En los primeros sistemas operativos, años 1950 y 60, cuando aún no se habían desarrollado metodologías de programación adecuadas, simplemente no había otra posibilidad. Más recientemente se han seguido diseñando sistemas operativos monolíticos por razones de rendimiento. Las diferentes implementaciones de UNIX son un ejemplo significativo.

[10] También conocido como modo protegido, modo sistema, modo *kernel*, o modo supervisor.

Por capas

Se proporcionan diferentes niveles de abstracción, o capas. Cada capa encapsula un conjunto de funciones y estructuras y proporciona una interfaz para la capa superior. La capa más interna, **núcleo** (o *kernel*)[11] del sistema operativo, utiliza y manipula directamente el hardware (memoria física, puertos de E/S…) y conmuta los procesos, siendo dependiente de la arquitectura soporte. La capa más externa proporciona al usuario-programador la interfaz de llamadas al sistema (canales, identificadores de proceso…). Las capas intermedias proporcionan sucesivas abstracciones, lo que se hace especialmente evidente, como veremos, en el sistema de ficheros. La estructura por capas simplifica el diseño del sistema operativo, permite un mantenimiento más sencillo y posibilita las modificaciones. El precio a pagar es la pérdida de rendimiento inherente a la implementación de toda abstracción de datos y funciones, lo que históricamente ha reducido estos sistemas prácticamente al ámbito de lo académico, con pocas excepciones (por ejemplo, VMS de Digital).

Cliente-Servidor

El sistema se divide en módulos independientes con entidad propia que soportan las diferentes funciones y que se comunican mediante paso de mensajes, de acuerdo al esquema cliente-servidor. Este es el enfoque de moda en la implementación de sistemas operativos en los últimos años. Debido a su carácter intrínsecamente distribuido, permite la implementación de las funciones del sistema en diferentes unidades de proceso. Windows NT[12] es un ejemplo de este enfoque.

Basados en micronúcleo

Una tendencia es la de reducir al mínimo el código del núcleo del sistema operativo (micronúcleo o *microkernel*), implementando en él aquellas funciones que estrictamente deben ejecutarse en modo privilegiado. El micronúcleo implementa los mecanismos de gestión básica de procesos, memoria, E/S y comunicación entre procesos. Cuando una aplicación requiere un servicio del sistema operativo, la llamada al sistema se ejecuta en el espacio de usuario mediante un código del cliente que cursa la petición a través del mecanismo de

[11] En los sistemas monolíticos, el término kernel se usa para referirse a todo lo que se ejecuta en modo privilegiado.

[12] Las últimas ediciones de Windows NT se han comercializado con los nombres de Windows 2000, Windows XP y Windows Vista. Estos no deben confundirse con Windows 95/98, ya que, aunque comparten interfaz de usuario, son muy diferentes en cuanto a estructura interna.

comunicación por paso de mensajes del micronúcleo. La petición se dirige a un servidor, que usualmente se implementa también en espacio de usuario sobre el micronúcleo. De acuerdo a este esquema, la interfaz de llamadas a un sistema operativo convencional, por ejemplo UNIX, se entiende como una aplicación más. Los micronúcleos proporcionan una gran flexibilidad: una aplicación desarrollada para un sistema operativo (por ejemplo UNIX) puede ejecutarse en un micronúcleo sobre el que se haya instalado como servidor la interfaz de llamadas al sistema de dicho sistema UNIX (compatibilidad a nivel fuente). Otras aplicaciones específicas pueden ejecutarse directamente sobre el micronúcleo (por, ejemplo, un sistema de gestión de base de datos o un videojuego). Mach 3.0 y Chorus son ejemplos de micronúcleos, desarrollados para soportar UNIX. El sistema operativo Mac OS X de Apple está basado en Mach 3.0.

1.4 El mercado de los sistemas operativos

Desde una perspectiva más cercana al mundo comercial, es preciso referirse a dos grandes grupos de sistemas operativos. En primer lugar, aquellos sistemas operativos que han sido diseñados por un fabricante para una arquitectura concreta con el objetivo de proteger sus productos (tanto software como hardware) de posibles competidores se denominan **propietarios**. El fabricante diseña el sistema operativo específicamente para la arquitectura y proporciona las actualizaciones necesarias. Incluso a veces las especificaciones (interfaz de llamadas al sistema) no se hacen públicas o se modifican constantemente, dificultando el desarrollo de aplicaciones por otros fabricantes. Se crea así un mundo cerrado que engloba la arquitectura, el sistema operativo propietario y las aplicaciones, que permite el control del fabricante sobre el mercado y establece grandes dependencias para los clientes. Algunos ejemplos de sistemas operativos propietarios son (o han sido) VMS de Digital para VAX, Apple Macintosh, y MS-DOS, Windows 95/98 y Windows NT de Microsoft para plataformas PC[13].

Con la aparición de UNIX (hacia 1970) nace una nueva filosofía: al estar escrito casi completamente en lenguaje de alto nivel, el sistema operativo es transportable a otras arquitecturas y por lo tanto también lo son las aplicaciones a nivel de código fuente. Sin embargo, aunque pueda hablarse de UNIX en general como un sistema no propietario, cada fabricante ha ido introduciendo sus propias modificaciones en la

[13] Aún y todo, hay importantes diferencias entre sistemas propietarios. Así, Microsoft tuvo el acierto en los años 1980 de *abrir* la plataforma software (interfaz MS-DOS) a otros desarrolladores. Esto fue en detrimento de Apple, entonces su gran competidor en el mercado doméstico, que partiendo con una ventaja tecnológica evidente cerró la plataforma a sus propios productos. A medida que la arquitectura PC ha ido conquistando mercados, los sistemas de Microsoft lo han hecho con ella.

interfaz de llamadas al sistema[14], de forma que más bien hay que referirse a diferentes familias de UNIX, no totalmente compatibles entre ellas (System V, BSD, AIX, ULTRIX, Solaris, Linux…). El ideal, un mundo de **sistemas abiertos**, con especificaciones públicas, aceptadas y estandarizadas, que permitan la transportabilidad plena de aplicaciones y usuarios, es un objetivo escasamente logrado. En este sentido, la especificación POSIX es un referente en el mundo UNIX. Los micronúcleos estaban llamados a servir como base para la coexistencia de aplicaciones heterogéneas, pero, como ya hemos comentado, su difusión es escasa. En su lugar hoy en día es habitual soportar compatibilidad ya sea mediante emulación del hardware, fundamentalmente por virtualización (p. ej., VMware, Virtual PC), ya mediante interpretación (Java Virtual Machine).

Hay que destacar un fenómeno que revolucionó el mercado del software y en particular de los sistemas operativos: la aparición espontánea de una comunidad de programadores que desarrollan **software libre**[15]. Internet constituye el medio necesario para la compartición y el intercambio ágil de ideas y código entre la comunidad. Como consecuencia, y así se ha demostrado ampliamente, se dinamiza la adaptación del software ante problemas particulares y el desarrollo de nuevos productos, y se corrigen errores y afinan versiones con gran agilidad. Organizaciones como GNU[16] otorgan licencia de copia, modificación y redistribución del software libre con la condición de que la nueva distribución incluya el código fuente[17]. Linux es un ejemplo hoy en día asentado de esta filosofía.

En la actualidad los sistemas operativos, más allá de su orientación original, han tenido que adaptarse a multitud de dispositivos que requieren gestionar recursos con capacidades nada despreciables de proceso y almacenamiento, como es el caso de teléfonos móviles, agendas electrónicas y otros dispositivos de consumo. A ello hay que añadir los **sistemas empotrados**, cada vez más presentes en nuestro entorno (electrodomésticos, automóviles, instalaciones industriales, robots, etc). En algunos casos, sistemas operativos convencionales se han adaptado a las restricciones de los dispositivos (de tamaño y potencia), como es el caso de Windows Mobile de Microsoft, iPhone OS de Apple o Palm OS; en otros casos se ha optado por desarrollos específicos, como es el caso de Symbian OS o de Android de Google. Los sistemas empotrados además de restricciones físicas presentan necesidades de **tiempo real**, en algunos casos críticas, que conducen a adoptar soluciones específicas. Ejemplos de sistemas operativos de tiempo real son VxWorks y QNX.

---

[14] En un principio, el código fuente de UNIX se distribuyó libremente.

[15] *Free software*. No confundir con *freeware*. Tampoco debe entenderse como software gratuito.

[16] http://www.gnu.org

[17] Esta licencia se denomina *Copyleft*.

1.5 Bibliografía

Los textos generales sobre sistemas operativos suelen presentar en el capítulo de introducción conceptos generales como los que hemos introducido aquí. Los más cercanos a nuestro enfoque son: [LIS93], [SAN05], y [TAN06] y [TAN08], que presentan una descripción de las estructuras del sistema operativo, y [FIN88], que define los conceptos de política y mecanismo y divide el estudio del sistema operativo en gestión del tiempo y gestión del espacio.

Los conceptos relacionados con la evaluación del rendimiento suelen definirse sobre la marcha en los capítulos que tratan sobre procesos y memoria, como también haremos nosotros.

Para mantenerse al día sobre el mundo comercial de los sistemas operativos hay que leer las publicaciones periódicas especializadas. Internet es un medio útil para conocer el estado de desarrollo de las nuevas propuestas.

1.6 Ejercicios

1 Introduciendo trazas en un sistema hemos medido que en un intervalo de tiempo de 1000 segundos, 870 segundos el sistema ha estado ejecutando código de programas de usuario, durante 60 segundos ha ejecutado diferentes algoritmos de gestión del sistema operativo (scheduler, gestión del espacio libre, etc), y el tiempo restante la CPU ha estado haciendo espera activa pese a que había programas esperando ejecutarse. Calcular la eficiencia temporal del sistema en ese intervalo.

2 Hemos ideado un sistema de almacenamiento con direcciones de 16 bits y direccionable byte a byte. (a) Calcular la capacidad de direccionamiento de este sistema. (b) Si modificamos el sistema definiendo como unidad de direccionamiento bloques de 1 Kbyte, ¿cuál es ahora la capacidad de direccionamiento? ¿Qué perdemos a cambio?

3 Un sistema de particionado fijo de 1 Mbyte se gestiona mediante un mapa de bits. (a) Calcular el tamaño de la unidad de asignación (que sea potencia de dos) de forma que la pérdida de eficiencia espacial por el mapa de bits no supere el 1%. (b) Si almacenamos objetos de un tamaño medio de 1 Kbyte, calcular una estimación de la fragmentación interna media por objeto. (c) Calcular la fragmentación interna (c1) para almacenar un objeto P1 de 20 bytes, y (c2) para almacenar un objeto P2 de 1 byte.

4 Un sistema de almacenamiento con un espacio de direcciones físicas de 256 Mbytes direccionables a nivel de byte se gestiona mediante particiones de tamaño variable de cualquier longitud. El espacio libre se representa mediante una lista encadenada de huecos como la de la Figura 1.2. El sistema compacta automáticamente cuando el número de huecos alcanza 1024. (a) Calcular el espacio necesario para almacenar la lista y expresar esta sobrecarga como pérdida de eficiencia espacial. (b) Si el sistema se gestionara por mapa de bits, ¿cuánto ocuparía éste?

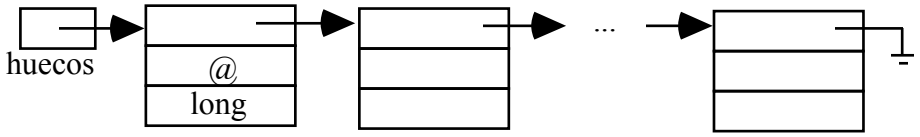

Figura 1.2. Lista encadenada de huecos.

5 En un sistema de gestión de memoria por particionado variable existen los siguientes huecos, en este orden: 10 Kb, 4 Kb, 20 Kb, 15 Kb, 7 Kb, 9 Kb, 12 Kb, 11 Kb. Si se tienen pendientes de cargar tres programas que vienen en el orden: A (12 Kb), B (10 Kb), y C (8 Kb), indicar cuál será el hueco que ocupen y la fragmentación consiguiente según la política de asignación de huecos sea first-fit, next-fit, best-fit o worst-fit.

6 Algunos sistemas trataban de eliminar la fragmentación externa compactando. Considérese un computador con 1 Mbytes de memoria que compacta, de media, una vez cada segundo y tarda 0,05 microsegundos en copiar cada byte (no se usa DMA). Se ha medido que, en el momento de la compactación, el tamaño medio de un hueco es el 40% del de una partición ocupada, y que el número de huecos tiende a ser el 50% del número de particiones. (a) Calcular la fragmentación externa total. (b) ¿Qué porcentaje del tiempo de proceso se usa para compactar?

7 Considérese un sistema de particionado variable donde el espacio libre se gestiona por una lista encadenada de huecos como la de la Figura 1.2. (a) Implementar la función `int asignar(int tamaño)`, que asigna un hueco de longitud `tamaño` utilizando un algoritmo next-fit y devuelve –1 si no hay un hueco suficientemente grande. Expresar el algoritmo en lenguaje C, introduciendo las definiciones necesarias. (b) Diseñar una función de liberar el espacio ocupado a partir de una dirección dada.