CAPÍTULO 8

SOFTWARE

Cuando compramos una computadora nos ocupamos de averiguar si viene sin sistema o si viene con Windows o Linux, antivirus, navegador de internet y otros "programas" que podemos necesitar para el uso cotidiano. Sabemos que sin esos "programas" ni siquiera podremos encender la computadora. También conocemos que esos "programas", o software, pueden ser adquiridos en los comercios del ramo, otros bajados gratuitamente de internet y que también se consiguen "pirateados" sin respetar legalmente los derechos de autor.

La rápida estandarización del hardware, con precios descendentes, extendió, en las últimas dos décadas, el uso de computadoras a pequeñas empresas y a los hogares, permitiendo la proliferación y estandarización del software con precios accesibles (incluso gratis) para las organizaciones menores y particulares.

Como hemos visto, para las organizaciones el software sigue siendo un factor fundamental como generador de ventajas competitivas, ya no tanto por la posibilidad de obtenerlo, sino por un uso adecuado y eficiente para el contexto en que se desenvuelve la organización. Las herramientas ahora no sólo están disponibles, sino que también son accesibles; pero si no se usan o se utilizan inadecuadamente, difícilmente ayudarán a la organización a progresar en este mundo competitivo y globalizado.

En este capítulo veremos las características del software, los distintos tipos, los lenguajes de programación que permiten desarrollarlos, los traductores que permiten pasar del código fuente al lenguaje de máquina, y las diferentes formas de licenciamiento del software.

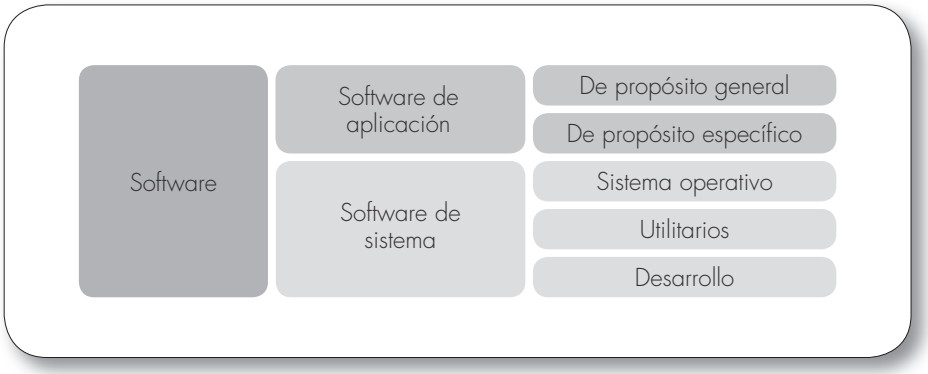
8.1 CONCEPTOS

Una computadora trabaja de acuerdo a un programa formado por una serie de instrucciones ordenadas en una secuencia predeterminada, siendo cada instrucción una orden específica en las que se descompone un trabajo. Es decir, un programa se integra de un conjunto de operaciones que han de ejecutarse. Cada operación es una orden que se imparte a la computadora para ser ejecutada. A través de las instrucciones se especifica lo que se va a hacer, cómo hacerlo y lo que se va a usar para llevar a cabo la operación, como hemos visto en el capítulo anterior.

Uno o más programas constituyen un software. En una computadora convivirán varios software de distintos desarrolladores destinados a diferentes necesidades.

El software puede ser clasificado según diferentes criterios, pero hay una división básica entre el software destinado a cubrir aplicaciones generales para cualquier usuario, o específicas de la organización y que hacen al giro de sus negocios o actividades (software de aplicación); y el software que se utiliza para la administración o gestión de los recursos físicos del hardware de las computadoras, o para tareas generales necesarias para el uso de dichos recursos (software de sistema o de base). La **Figura 8.1** presenta esta clasificación.

Figura 8.1
Software



A su vez, todo software debe ser desarrollado y para ello se requieren lenguajes de programación que permitan definir los pasos o instrucciones para cumplir con una tarea. Para una mayor facilidad, esos lenguajes tratan de ser cada vez más amigables para el ser humano, lo que los aleja del lenguaje binario de máquina, que es el único que puede ejecutar un procesador. En consecuencia, los traductores (compaginadores, compiladores, intérpretes y entornos) permiten pasar de los lenguajes simbólicos usados para la codificación, al lenguaje de máquina.

La propiedad intelectual y los derechos de autor incluyen el software y, por lo tanto, no tener en cuenta este aspecto puede generar importantes inconvenientes legales y económicos a la organización. Un reciente estudio realizado por Prince & Cooke revela que sólo el 9,7% de las empresas sabe que su información valiosa está en riesgo a causa de la piratería y que el 70,8% de las pequeñas y medianas empresas en Argentina reportaron fallas asociadas al uso de software ilegal que redujeron su productividad.

8.2 SOFTWARE DE APLICACIÓN

El software de aplicación es un conjunto de programas concebidos o creados para atender trabajos generales o específicos del usuario, referidos al cumplimiento de sus diversos objetivos. En cambio, el software de sistema o de base, actúa como apoyo para que

podamos usar la computadora, pero cualquier actividad que queramos realizar desde un punto de vista funcional implicará la utilización de un software de aplicación.

La diferencia fundamental de este conjunto de programas con los que consideramos más adelante, integrando el software de sistema o de base, radica en el hecho que usualmente estos últimos nos permitirán usar la computadora, administrarla, mantenerla, pero en ningún caso realizar las tareas cuyo objetivo dio origen a la incorporación de la computadora.

Todas las aplicaciones vistas en la Parte II forman parte del software de aplicación.

El software de aplicación necesita parte del software de sistema para ejecutarse en la computadora. Sin embargo, existe la posibilidad de que los programas sean ejecutados sin software de base, hecho que puede acarrear gran complejidad en el desarrollo de los mismos; por esta razón, expresamos que el software de aplicación necesita del software de base para ser ejecutado. En la **Figura 8.2** se muestra que el usuario se relaciona con el software de aplicación que, a su vez, utiliza los servicios del software de sistema (sistema operativo) para acceder al hardware.

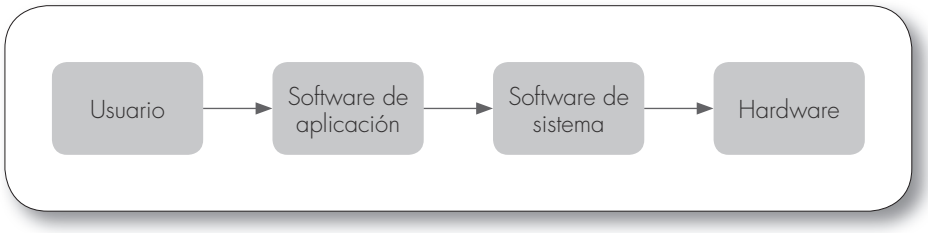


Figura 8.2
Relación usuario - software - hardware

El software de aplicación puede ser dividido en:

- De propósito general (tareas comunes para todo tipo de usuarios)
- De propósito específico (tareas específicas del usuario)

Ejemplos de software de aplicación de propósito general son los navegadores de internet, correo electrónico, procesadores de texto, planillas de cálculo electrónicas, software para presentaciones, software para trabajo en grupo (*groupware*), entre otros. Las aplicaciones ERP, CRM, SCM, BPM, HCM y otras vistas en la Parte II y otras aplicadas a la educación, al entretenimiento, arquitectura, ingeniería, calidad, medicina, etcétera, son ejemplos de software de aplicación de propósito específico.

Las formas para obtener un software de aplicación pueden ser:

- *Confeccionados especialmente para un usuario determinado:* esta alternativa ofrece la oportunidad de introducir en el software las características propias de ese usuario determinado. Es decir, que los sistemas desarrollados "a medida" se generan teniendo en cuenta las particularidades que esa aplicación tiene para ese usuario.

  El desarrollo puede ser realizado internamente o contratado a una organización externa. En ambos casos, tanto la propiedad del software desarrollado como el código fuente y la documentación técnica interna, deben quedar en manos de la empresa contratante.
- *Incorporando paquetes pre planeados:* los paquetes también se orientan a aplicaciones específicas de los usuarios, pero son concebidos en forma estándar. Se generan tomando en cuenta las características propias de la aplicación para satisfacerla en sus requerimientos estándares, generales, sin tener en consideración a usuarios en particular, con el fin de comercializar múltiples copias.

Se trata de sistemas que, desde el punto de vista de los requerimientos de una aplicación específica de un usuario determinado, podrían resultar menos adaptados que los desarrollados especialmente, debido a que tienen en cuenta menos características de cada usuario en particular. No obstante, desde el punto de vista de los requerimientos generales de la aplicación considerada en sí misma, son sistemas mucho más flexibles que los desarrollados para la satisfacción de requisitos de usuarios en forma particular.

Los paquetes de gestión (ERP), los de relacionamiento con los clientes (CRM), los software integrados de oficina (MS Office, Open Office, StarOffice de Sun, entre otros), de correo electrónico, navegadores de internet y otros de uso corriente, son también ejemplos de software de aplicación pre planeados.

Frente a una necesidad de procesar por computadora una aplicación con estos sistemas, el usuario tiene la posibilidad de adquirir los software ya desarrollados en forma estándar, reuniendo las características comunes de la aplicación a la que se refieren, aunque ninguna característica particular del usuario que lo adquiere.

En resumen, diremos que el modelo de negocios, tomado en cuenta en el desarrollo de un software confeccionado para el usuario, es la necesidad específica de ese usuario. En cambio, en los sistemas pre planeados, el modelo tomado en cuenta resulta ser la conjunción de los requerimientos generales que se persigue satisfacer con dicha aplicación, independientemente del usuario particular.

Por supuesto que es posible una estrategia intermedia, tomando lo que me sirve de un paquete (a un costo muy inferior al desarrollo) y desarrollando complementos que permitan "personalizar" la aplicación a las necesidades de los usuarios, o bien, contratando al desarrollador del paquete para que "personalice" directamente el propio paquete. Probablemente, en estas estrategias intermedias se pueda encontrar la mejor combinación de menores costos iniciales (de única vez) y operativos (de todos los días).

El esquema tradicional es que el software de aplicación se ejecuta en la infraestructura de TICs de la organización (equipamiento y software en las oficinas propias). Sin embargo, existe una alternativa diferente que se denomina ASP (*Application Service Provider* o proveedor de servicio de aplicación). Ya sea hecho "a medida" o paquete, el software de aplicación tiene como característica estar basado en Web y tiene bajo costo de inversión inicial, porque reduce en gran medida las necesidades de infraestructura (servidores, licencias de software de sistema, personal especializado, etcétera) ya que se ejecuta en los servidores del proveedor (que mantiene la infraestructura); sólo requiere el uso de un navegador de internet. El costo normalmente se cobra por transacciones o por valores fijos mensuales o anuales. De acuerdo a como sea, puede tener un costo variable más alto, que incluso supere los ahorros de los costos iniciales (a lo largo del tiempo). Por otro lado, existe una mayor dependencia del funcionamiento de la conexión de internet y de la infraestructura del proveedor, ambos aspectos fuera de nuestro alcance.

En la Parte IV analizaremos las ventajas y desventajas de adquisición y desarrollo, y sus diferentes alternativas.

8.3 SOFTWARE DE SISTEMA

Para definir el software de sistema o de base lo dividiremos en tres grandes grupos:

- Sistema operativo
- Software para desarrollo
- Utilitarios

8.3.1 Sistema operativo

Los sistemas operativos son conjuntos de programas concebidos para efectuar la administración o gestión de los recursos físicos de la computadora. La filosofía que anima su desarrollo es que la computadora debe trabajar lo más continua y eficientemente como sea posible, y para ello el sistema operativo efectuará la administración de los recursos en proceso.

El desarrollo y evolución alcanzados en materia de sistema operativo llevó a simplificar el uso de la computadora, al punto que ya es utilizada, sin dificultades, por niños que aún no han comenzado su educación formal.

Hubo y hay muchos sistemas operativos, pero los más conocidos y difundidos son Windows, Linux, UNIX, Mac OSX y Chrome OS. Algunos sistemas operativos han sido desarrollados especialmente para ser servidores de red, como Novell Netware o Windows Server. En una red podrán convivir distintos sistemas operativos, como por ejemplo Windows para las PCs de usuario, un servidor general con Windows Server y otro servidor de base de datos UNIX.

Es frecuente que a un sistema operativo se lo denomine también "plataforma". Cuando un software funciona en distintas plataformas se lo denomina multiplataforma (aún cuando haya una versión distinta para cada una de las plataformas). Para que un software "corra" en una plataforma, por un lado su lenguaje de máquina debe poder ejecutarse en el procesador (hay procesadores compatibles aún de distinto fabricante, como es el caso de Intel y AMD, y también procesadores incompatibles, aún del mismo proveedor) y por otro, debe poder relacionarse y utilizar los servicios del sistema operativo. Por ejemplo, un software que puede ejecutarse en una PC normal puede correr en Windows y no correr en Linux, y viceversa.

En una misma computadora pueden coexistir más de un sistema operativo, ya sea en forma real como en forma virtual (si se utiliza un software de virtualización). Si se trata de sistemas operativos reales, sólo uno de ellos queda en ejecución y para utilizar otro es necesario reiniciar la computadora y seleccionarlo. En cambio, si se utiliza virtualización, todos los sistemas operativos se encuentran activos al mismo tiempo y no es necesario reiniciar para trabajar sobre alguno de ellos. Cada uno de ellos se puede considerar como un equipo virtual diferente. Esta característica es sumamente provechosa en el caso de los servidores, aunque también cómoda en cualquier computadora.

En el siguiente punto se analizarán en detalle los objetivos, características y funcionamiento de los sistemas operativos.

8.3.2 Software para desarrollo

Para el desarrollo de software existe, asimismo, un conjunto de softwares específicos que permiten facilitar el proceso de construcción. Se utilizan lenguajes de programación que resultan más sencillos de manejar por los desarrolladores, pero que deben ser traducidos al lenguaje binario de máquina para poder ser efectivamente ejecutados. En el Punto 8.5 veremos los lenguajes de programación y en el 8.6 los traductores (compaginadores, compiladores, intérpretes y entornos), que permiten pasar de los lenguajes simbólicos usados para la codificación, al lenguaje de máquina. Otro ejemplo de este tipo de software son los sistemas de administración de base de datos que permiten definir, visualizar, modificar, copiar y manejar, entre otras funciones, bases de datos (parte de sus funcionalidades podríamos clasificarlas como utilitarios). En el Capítulo 10 veremos este tema con mayor profundidad. También existen muchas herramientas dedicadas al desarrollo de software, como aquellas gráficas para documentación técnica, para prueba, herramientas CASE (*Computer Aided Software Engineering*, es decir, ingeniería de software asistida por computadora), etcétera.

8.3.3 Utilitarios

Los programas utilitarios son usados para cumplir un número de funciones que, por estándares y repetitivas, resultan necesarias para cualquier usuario en tareas relacionadas con el uso, diagnóstico y mantenimiento de la computadora.

Muchos utilitarios vienen acompañando al sistema operativo, muchos otros son de uso gratuito y pueden ser bajados de internet y otros son fácilmente adquiribles en cualquier negocio del ramo o en la red. Pero difícilmente una organización desarrolle software de sistema, obviamente salvo que sea una organización que se dedique a este tipo de desarrollos.

Como ejemplos podemos citar los antivirus, *firewalls*, *antispyware*, utilitarios de diagnóstico, de información de hardware, monitores del hardware, monitores de red, la calculadora, el liberador de espacio de disco, el explorador, inicializador de discos, el reorganizador de espacio en disco, el editor de textos (como por ejemplo, el block de notas), reproductores de audio y video, control y reparación de errores en la superficie de disco, restauración de sistema, etcétera.

Veamos un par de ellos muy utilizados.

a) Explorador

El directorio (denominado también FAT, del inglés *File Allocation Table* o tabla de alocación o asignación de archivos) es básicamente un archivo más dentro de una unidad de discos (o medio equivalente como *diskette*, CD, DVD o *pendrive*) que contiene además de la identificación o nombre del disco, una serie de datos de los archivos lógicos que están grabados en ese volumen así como también la cantidad de espacio disponible y su ubicación. Es decir, es el índice del contenido del volumen (por eso también se usaba el nombre de VTOC, del inglés *Volume Table of Contents* o tabla de contenidos del volumen). A su vez, en una estructura de tipo árbol, pueden existir subdirectorios (o carpetas), que, a su vez, pueden contener archivos y otros subdirectorios o carpetas. Cuando se dice que la FAT tiene una determinada cantidad de bits (por ejemplo 32), se indica que 2 elevado a ese número, es la máxima cantidad de unidades de asignación que se pueden direccionar en una unidad lógica de disco (recordemos que en un disco físico podemos tener uno o más discos lógicos o particiones).

En general, los datos más usuales que contiene de cada archivo lógico son el nombre del archivo, la cantidad de espacio que ocupa, la/s dirección/es del archivo (ubicaciones o extensiones), la fecha de creación, la fecha de modificación y atributos del archivo, entre otros.

Este utilitario permite visualizar el contenido de directorio y subdirectorios, así como de otros equipos conectados en red y características de configuración (panel de control).

También nos permitirá eliminar un archivo (suprimiendo los datos de dicho archivo del directorio, desasignando el espacio ocupado por dicho archivo lógico y colocándolo como espacio disponible) o renombrar un archivo (que consiste simplemente en cambiar el nombre del archivo en la entrada del directorio del soporte por uno nuevo).

Este utilitario es uno de los más usados, ya que también nos permite buscar, copiar archivos, ver sus propiedades, abrirlos o ejecutarlos desde su interfase, ver el espacio libre, etcétera.

b) Reorganizador de espacio en disco (desfragmentador)

Es frecuente que, con motivo de sucesivas incorporaciones y eliminaciones de archivos de un volumen (por ejemplo disco), se produzcan una cantidad considerable de pequeñas extensiones (cierta cantidad de pistas o sectores) entre los distintos archivos lógicos.

Además, la actualización de los datos de los archivos harán que el mismo comience a estar ubicado en diferentes segmentos no contiguos del soporte. Estos espacios, aún cuando puedan ser manejados por las rutinas de asignación automática de espacio del sistema operativo, generalmente producirán una pérdida de tiempo tanto en la grabación del archivo en esas extensiones como también en su lectura, como hemos visto en el capítulo anterior. Además, obligarán a guardar las direcciones de dichas extensiones en el directorio general del volumen haciendo más lento también encontrar la ubicación de un archivo lógico en el directorio (dado que las extensiones adicionales de un archivo también ocupan lugar en el directorio). Por lo tanto, cuando el espacio ocupado y libre en disco se encuentra muy fragmentado, se hace necesario compactar (desfragmentar) los archivos y reorganizar los espacios. Recuerde que el movimiento de la cabeza del disco es el tiempo más importante de una operación de lectura o grabación. Cuando hay bastante fragmentación, la ejecución de este proceso permitirá lograr una mejora en el rendimiento del equipo.

La función de este utilitario, también denominado desfragmentador de disco, será entonces la de ir copiando los archivos lógicos dentro del volumen sin dejar espacios libres entre ellos. Por otro lado, si un archivo está segmentado en diferentes ubicaciones no contiguas, junta los diferentes segmentos del archivo, colocándolo en un sólo segmento contiguo. Esta tarea involucra, además, la actualización del directorio general del soporte en lo que hace a las ubicaciones de los archivos lógicos y al nuevo espacio libre (que será uno solo al final de todos los archivos lógicos, reduciendo también la cantidad de registros en el directorio necesarios para identificar las áreas libres).

8.4 SISTEMAS OPERATIVOS

Los sistemas operativos son conjuntos de programas concebidos para efectuar la administración de los recursos de la computadora. Algunos de ellos se encuentran residiendo permanentemente en la memoria principal (luego de efectuada la carga inicial al prender la computadora) mientras la computadora esté encendida. Otros residen en memoria, sólo cuando se los necesita ejecutar, encontrándose almacenados en unidades de memoria secundaria (por ejemplo, discos). A los primeros se los conoce como: residente, supervisor, monitor, ejecutivo, kernel o núcleo. A los segundos, como transientes.

La creciente complejidad de los sistemas operativos encuentra su causa en la también creciente complejidad que manifiesta el hardware.

El sistema operativo tiene dos objetivos básicos:

- Facilitar el uso de la computadora, proporcionando servicios para la ejecución de programas, es decir, obtener automáticamente el programa apropiado y administrar los recursos de los procesos en ejecución.
- Actuar como entorno de la aplicación, en el cual el programa es ejecutado, administrando los recursos de una manera eficiente (supervisar las operaciones de la computadora, determinar la asignación del procesador, administrar el uso de la memoria principal, dirigir el desenvolvimiento de las operaciones de entrada/salida y del acceso a archivos).

Los componentes que conforman un sistema operativo materializan, entre otras, las siguientes tareas en la ejecución de una aplicación:

- Carga de programas y componentes
- Administración y manejo de las unidades del hardware
- Administración y manejo de datos
- Comunicación de programa a programa

- Interfase hombre/máquina/sistema de aplicación
- Supervisión de la ejecución de los diferentes programas
- Alocación de programas/datos en la memoria
- Manejo de interrupciones
- Mantenimiento de flujo constante de trabajo a la computadora
- Tareas de comunicación de datos

Las funciones de un sistema operativo son:

- Carga inicial de los componentes residentes en la memoria principal
- Administración de memoria principal
- Administración del/los procesador/es
- Administración de los dispositivos de entrada/salida
- Administración de los procesos a ser ejecutados
- Administración de datos

8.4.1 Multiprogramación. Multiprocesamiento

Cuando se ejecutan varios programas o tareas en forma concurrente, surgen importantes complicaciones para el uso del hardware. En este punto trataremos de explicar esta forma de trabajo bajo dos modalidades, conocidas como multiprogramación y multiprocesamiento, presentes en cualquier sistema operativo moderno. Si bien estos conceptos tienen cierta complejidad técnica, se presentarán de forma simplificada.

a) Multiprogramación

La multiprogramación consiste en el manejo casi simultáneo de dos o más programas independientes, intercalando su ejecución y compartiendo tiempos del procesador. El control de la ejecución de esta intercalación lo realiza el sistema operativo.

Significa entonces que, por medio de la multiprogramación, se efectúa la administración de la ejecución en paralelo de dos o más programas que residen simultáneamente en la memoria de la computadora.

Algunas características generales que podemos enunciar son las siguientes:

- *Intercalación:* ya que más de un programa se encuentra cargado en la memoria principal en condiciones de ejecutarse, ejecutándose o demorados. Pero todos ellos compartiendo tiempos de procesador y asignaciones de memoria.
- *Instantaneidad:* ya que se simula trabajar como si existiera un sólo programa cargado de ejecución instantánea.
- *Independencia:* se trata de distintos programas, con distintas asignaciones de memoria y de dispositivos de hardware.

Los sistemas operativos que trabajan en multiprogramación tienen forma de proteger la memoria de trabajo de cada programa. Así se evita que, por ejemplo, si en un momento determinado se encuentran dos programas ejecutándose, por error uno de ellos almacene datos en direcciones ocupadas por el otro y se destruyan instrucciones o datos de este último.

Como quedó establecido más arriba, el objetivo principal de la multiprogramación es el aprovechamiento del procesador, permitiendo que varios programas o diferentes aplicaciones se estén ejecutando, intercalándose y compartiendo tiempos. De esta manera, se minimizan los "tiempos de espera" en que el procesador se encuentra "inactivo". Por ejemplo, cuando un programa espera datos que deben buscarse en un disco, el sistema operativo asigna el procesador a otra tarea o programa. Paralelamente a este objetivo se debe desarrollar otro, que es evitar la interferencia mutua entre dichos pro-

gramas o aplicaciones. Uno de los métodos que brinda óptimos resultados es convertir las direcciones utilizadas en un programa en direcciones lógicas y aplicar estas últimas sobre direcciones físicas. Así, aparentemente, varios programas se referirán a la misma dirección, pero el control que el sistema operativo ejerce permite que se ejecute asegurando que cada uno de ellos tendrá acceso a una dirección física diferente. Metodologías precisas de este manejo lo constituyen el mecanismo de trabajo de asignación automática de memoria y memoria virtual, que veremos más adelante.

b) Multiprocesamiento

Hasta aquí hemos desarrollado el tema, indicando la posibilidad de trabajar compartiendo tiempos de proceso y, casi en paralelo, efectuando el aprovechamiento de la UCP con varios programas y un solo procesador. Pero ampliar el concepto de un trabajo real en paralelo y a una ejecución simultánea de programas nos obliga necesariamente a aplicar más de un procesador.

De manera que el problema a resolver por el sistema operativo consistirá en asignar una cantidad de procesadores N a una cantidad de programas M, donde generalmente $N < M$, pero sabiendo que $N > 1$. En este caso estamos en presencia del multiprocesamiento y, como en multiprogramación, el sistema operativo deberá asignar cada procesador a los distintos programas sabiendo que contamos con más de un procesador.

Cuando un programa termina o se detiene, uno de los procesadores queda disponible y se podrá asignar a otro programa o proceso.

8.4.2 Funciones

Como hemos visto anteriormente, la función de un sistema operativo expresada genéricamente es la de administrar los recursos de la computadora. En este punto veremos cuáles son esos recursos a ser administrados, y cuáles son las estructuras que habitualmente adoptan los distintos sistemas operativos.

El sistema operativo tiene a su cargo la administración de cinco elementos principales:

- Memoria principal
- Procesador/es
- Dispositivos de entrada/salida
- Procesos a ser ejecutados
- Datos

Varias veces hemos hecho mención a que no todo el sistema operativo está en memoria permanentemente. La parte que reside siempre en memoria principal durante la ejecución de los distintos programas recibe las denominaciones de residente, núcleo, supervisor, ejecutivo, monitor, *kernel*, etcétera. Nosotros utilizaremos los nombres de residente o supervisor para referenciar esta parte del sistema operativo.

Una vez encendida la computadora tendremos que cargar el supervisor en memoria para poder comenzar a ejecutar los distintos programas; pero en este caso, por ser el primer programa que será traído a la memoria ¿cuál será el programa encargado de traer el supervisor a memoria y de cuál dispositivo lo tomará?

En todos los sistemas operativos existe un programa muy especial que no cumple una función de administración de recursos y que tiene como única misión traer a memoria al supervisor, y se lo considera formando parte del sistema operativo. Este proceso recibe normalmente el nombre de *booteo*.

Este programa, una vez que se encuentra en memoria, comenzará a ejecutarse cumpliendo una serie de tareas que difieren de acuerdo al sistema operativo. Por ejem-

plo, en las PCs permite la modificación de los datos del BIOS. Finalmente, verificará cuál es el dispositivo donde deberá buscar al supervisor para traerlo a memoria. Esta indicación puede ser dada manualmente por el operador (si, por ejemplo, tenemos más de un sistema operativo en el disco), o bien, por medio del BIOS.

Una vez cargado el supervisor en memoria, este programa desaparece; en consecuencia diremos que es un programa transiente del sistema operativo.

Si bien, por lo general este programa sólo se ejecutará una vez hasta que el equipo sea apagado, puede haber razones para necesitar "bootear" nuevamente. Entre ellas podemos citar: cambiar de sistema operativo cuando la computadora está "colgada" o un programa no responde, restaurar el sistema frente a errores desconocidos, etcétera. Esta operación puede ser realizada a través del teclado, del *mouse* o por una tecla de la computadora (*reset*).

a) Administración de memoria

Desde los primeros sistemas operativos se han utilizado distintos métodos para la administración de la memoria, de acuerdo a las características de las computadoras y de las modalidades de procesamiento; la evolución de las técnicas de utilización de la memoria, junto con el aumento del tamaño de las memorias reales, han logrado superar la limitación que la memoria real representaba en el pasado. Cualquiera sea la forma que se utilice, será el supervisor del programa el que se encargue de las tareas necesarias para la administración de la memoria.

Sería interesante ver la historia de los distintos métodos, pero a la vez, demasiado detallado para los propósitos de esta publicación, así que sólo analizaremos la memoria virtual, ya que es la forma que, con algunas variantes, utilizan actualmente la gran mayoría de los sistemas operativos en uso.

a.1) Memoria Virtual

A pesar de que la memoria principal es cada día menos costosa (al mismo tiempo, el software cada vez utiliza más memoria), y que los equipos y sistemas operativos pueden gestionar un mayor espacio de direcciones reales de memoria, todavía es un recurso que debe ser usado eficientemente, porque tiene mucha importancia en la *performance* total de una computadora.

Eliminar las restricciones de memoria es un paso importante para lograr implementar software de aplicación a un costo menor y en menor tiempo. En consecuencia, se debe intentar lograr un mayor espacio de memoria pero sin ampliar la memoria real.

Teóricamente el requerimiento planteado puede ser satisfecho proveyendo un espacio de direcciones (un espacio de memoria) que será soportado utilizando dispositivos de acceso directo (en un disco), y además hardware y/o software para realizar las conversiones de las direcciones virtuales a las reales de la memoria. Además, como veremos más adelante, el hardware y/o software de conversión de direcciones ofrece capacidades funcionales que una mayor cantidad de memoria real no puede proveer.

La memoria virtual es un espacio de direcciones virtuales en una unidad de almacenamiento externo de acceso directo (disco magnético), cuyo tamaño máximo está determinado por el esquema de direccionamiento del computador. Si cada byte tiene una dirección distinta, la cantidad de bytes que pueden ser referenciados dependerá de la cantidad de bits que se utilicen para expresar una dirección. Por ejemplo: con 1 bit sólo pueden ser referenciados 2 bytes, el 0 y el 1; con 8 bits se podrán direccionar 256 bytes, con 16 serán 65.536 y con 24 serán 16.777.216 bytes. El resultado surge de elevar 2 a la cantidad de bits utilizados. Para hacer un ejemplo más fácil, si utilizáramos cuatro dígitos decimales para los números de puerta de una calle de la ciudad, podríamos tener números de 0 a 9.999, es decir, que ninguna calle podría tener más de cien cuadras.

En cambio, el almacenamiento que puede ser directamente accedido por el procesador se denomina memoria real. Hay pues, una distinción entre el espacio de direcciones de la memoria virtual y el de la memoria real. El espacio de la memoria real es un conjunto de ubicaciones de memoria en el cual las instrucciones y datos de un programa deberán ser ubicados para su procesamiento. El número de direcciones en esos dos espacios no necesariamente es el mismo, a pesar de que ambos empiezan con la dirección 0 y tienen direcciones consecutivas.

Cuando no existe memoria virtual, no hay diferenciación entre el espacio de direcciones y la memoria real; el espacio de direcciones que puede ser usado en los programas tiene idéntico tamaño al espacio de memoria real disponible. En cambio, si utilizamos memoria virtual, el espacio de direcciones utilizable por los programas es aquel determinado por el tamaño de la memoria virtual implementada (automática o manualmente) y no el espacio de direcciones provisto por la memoria real disponible. Los programas se refieren a los datos e instrucciones por la dirección de memoria virtual, sin conocer la ubicación física de memoria real.

Se denomina memoria virtual porque representa una imagen de memoria en lugar de una memoria principal física. Es importante recalcar que, dado que la memoria virtual no existe como una entidad física de memoria principal, las instrucciones y datos de un programa referenciados por direcciones virtuales deben ser contenidos en alguna ubicación física de memoria real para ser ejecutados; es decir, que para su efectiva ejecución, los datos e instrucciones correspondientes deben ser llevados de la memoria virtual a la memoria principal (no es necesario que sea todo el programa).

La parte residente del sistema operativo no integra la memoria virtual y se encontrará alojada permanentemente durante la ejecución de los distintos programas, en ubicaciones contiguas de la memoria real.

Los contenidos de la memoria virtual están divididos en porciones o secciones de tamaño fijo (por ejemplo: 1 K, o 2 K o 4 K). Asimismo, ya hemos visto que no es necesario que todo el programa esté en memoria en un momento dado. El programa estará completo en la memoria virtual, pero en la memoria real sólo estarán algunas secciones o páginas del mismo que irán cambiando a lo largo de su ejecución. El espacio de direcciones de la memoria virtual, que estará contenido en dispositivos de acceso directo, corresponde a los programas que se están ejecutando.

A su vez, la memoria real también está dividida en secciones de igual tamaño que las páginas. Cuando se debe ejecutar un programa, éste es traído a la memoria virtual, y las instrucciones y datos del programa son transferidos entre la memoria virtual y la real de a una sección por vez, durante la ejecución del programa. Inicialmente se cargará la primera página y luego según las necesidades. Una sección del programa será llevada a la memoria real sólo cuando es requerida, es decir, cuando una dirección de memoria virtual que corresponde a esa sección, es referenciada por otra sección que se está ejecutando en la memoria real. Por otro lado, una sección que está en memoria real sólo será reescrita en la memoria virtual cuando la memoria real asignada a esa sección sea requerida por otra sección del mismo o de otro programa, siempre que no se esté ejecutando y, además, si ha sido modificada. Si no ha sido modificada, simplemente se asignará ese bloque de memoria real a la sección que requiere memoria real para su ejecución, sin ser reescrita en la memoria virtual, dado que el contenido de la página es el mismo tanto en la memoria real como en la virtual (no ha sido modificada).

En general, se controla la actividad de las secciones de todos los programas que se están ejecutando a fin de mantener, en la medida de lo posible, en memoria real a las secciones más activas, dejando las menos activas en la memoria virtual.

Dado que cada sección o página puede ser ubicada independientemente en cualquier bloque de memoria real, no hay necesidad de que un programa ocupe bloques contiguos de la memoria real.

El hardware de traducción de direcciones o la función interna de *mapping* son los mecanismos por los cuales se pueden traducir las direcciones de memoria virtual en direcciones de memoria real durante la ejecución de las instrucciones. El sistema operativo mantiene distintas tablas que indican, entre otros datos:

- Cantidad de memoria virtual implementada.
- Secciones que están presentes en la memoria real.
- Direcciones indicando la ubicación en memoria real de cada una de dichas secciones.
- Elementos de juicio para determinar qué secciones se tratarán de dejar en la memoria real y cuáles no, o eventualmente, qué sección será desplazada cuando otra sección de memoria virtual deba ser llevada a memoria real.

Como hemos visto, el manejo de páginas implica conocer concretamente si una página se encuentra o no en la memoria real y, por otro lado, qué página se desplazará cuando la memoria real esté completa y se deba traer una nueva página.

La tabla de cada programa tiene el formato de la **Tabla 8.1**.

Tabla 8.1 | Tabla de páginas de un programa

Número de página	Indicador memoria real	Indicador páginas modificadas	Indicador de uso	Dirección memoria real
1	1	1	X	Dirección 1
2	0	0		
3	0	0		
4	1	0	Z	Dirección 2
5	0	0		

La primera columna indica el número de página del programa. La segunda es una marca que indicará si se encuentra en memoria real (código 1) o no (código 0). La tercera indicará, para las que se encuentren en memoria real, si han sido modificadas (código 1) o no (código 0). La cuarta columna le dará al sistema operativo otros elementos de juicio para decidir qué página será desplazada cuando se deba traer otra página a memoria real. Finalmente, la última columna nos dará la dirección de memoria real de aquellas páginas que se encuentren en ella.

Cada vez que se haga referencia a una dirección virtual al ejecutarse el programa, se buscará la tabla de páginas descrita, como veremos más adelante al analizar la forma de convertir las direcciones virtuales en reales.

La segunda columna nos dirá si ya se encuentra en memoria real. En caso negativo se originará una interrupción por página faltante, ordenándose la ejecución de una operación de entrada/salida, destinada a traer la página, previo análisis de la ubicación de la misma. En cambio, si ya estuviera en memoria, se tomará la dirección real de la página que sumada al desplazamiento dentro de la página dará la dirección de memoria real buscada. Además, se deberán actualizar las columnas 3 y 4. El indicativo de página modificada (columna 3) sólo será puesto en 1 cuando algún byte de la página sea modificado. De esta manera, cuando esta página sea desplazada de memoria real, se analizará este código y si es 1 se deberá grabar el contenido de la página (en memoria real) a la memoria virtual; en caso contrario, no será grabada, pues, si no se modificó ningún byte, será igual a la página de la memoria virtual. Este indicativo también será utilizado

para determinar qué página será desplazada de la memoria real, ya que será preferible desplazar una página cuyo código sea 0, debido a que no requerirá su grabación en la memoria virtual.

El contenido de la columna 4, indicador de uso, variará según el sistema operativo. Cualquiera sea el método utilizado, el criterio es tener datos o llevar la cuenta de la cantidad de veces que ha sido utilizada la página y, en consecuencia, cuando sea necesario se desplazará la que registre un menor uso.

Al utilizar memoria virtual, las direcciones virtuales se convierten en direcciones reales de memoria de diferentes maneras.

La más sencilla es aquella que direcciona una página del programa y un desplazamiento dentro de la página. Si consideramos que un programa está dividido en páginas de tamaño fijo, toda dirección podrá ser expresada como número de página y desplazamiento dentro de ella (el valor que puede asumir el desplazamiento será 0 hasta el tamaño de la página menos 1).

La **Figura 8.3** muestra la forma de conversión de direcciones virtuales en reales.

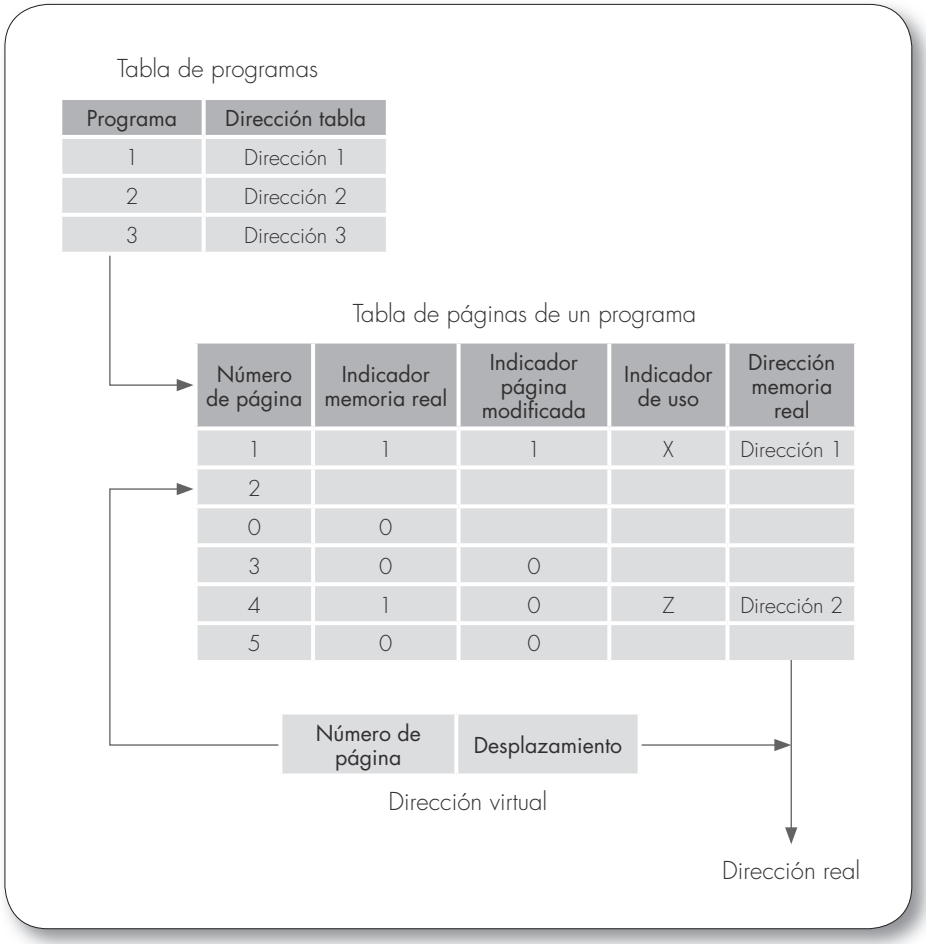


Figura 8.3
Conversión de dirección virtual en dirección real

Cada vez que haya que convertir una dirección virtual en real se accederá a una tabla (donde están todos los programas en ejecución) que indicará, para cada uno de ellos, dónde comienza la tabla de páginas de **Tabla 8.1**. Con esa dirección y el número de página de la dirección virtual a convertir, se buscará en la tabla del programa y se deter-

minará, mediante el indicativo de presencia en memoria real, si la página referenciada se encuentra o no en memoria real. En caso afirmativo se tomará la dirección de comienzo de la página que figura en la última columna de la tabla, se le sumará el desplazamiento dentro de la página (de la dirección virtual a convertir) y el resultado nos dará la dirección de memoria real, como se grafica en la **Figura 8.3**.

Otra forma de direccionamiento virtual consiste en subdividir la dirección en tres niveles: número de segmento, número de página dentro del segmento y desplazamiento dentro de la página. Cada segmento puede tener como máximo un determinado número de páginas (determinado por la cantidad de bits para expresar el número de página dentro del segmento).

La *performance* de la computadora puede estar directamente afectada por la cantidad de tiempo utilizado transfiriendo secciones de la memoria virtual a la real y viceversa (esta actividad se denomina paginado). Sin embargo, se puede lograr un buen rendimiento si cada programa en ejecución logra suficiente memoria real de tal manera que las transferencias se mantengan en un nivel razonable. Consideremos que, trabajando en multiprogramación, mientras un programa esté esperando que una sección sea llevada a memoria real, otro se puede estar ejecutando y a estos efectos, la transferencia de una página es equivalente a efectuar una operación de entrada/salida.

Cuanto mayor sea la actividad de paginación, mayor será la probabilidad de una menor *performance* del computador. Es por ello que, si tenemos varias aplicaciones abiertas, aún cuando aparentemente no estén trabajando, la respuesta del equipo puede ser más lenta.

Un balance adecuado entre memoria virtual y real busca el equilibrio de tiempos de paginado y espera para procesos, a los efectos de mejorar la *performance* general del equipo.

Aún cuando la memoria virtual puede ser de muchos *gigabytes*, para determinar su tamaño más conveniente hay que tener en cuenta distintos factores como:

- El tamaño de la memoria principal.
- La velocidad del dispositivo de acceso directo que va a contener la memoria virtual y el canal al cual está conectado, y su grado de utilización.
- La velocidad de procesador/es.
- Las características de los programas que se ejecuten concurrentemente.

b) Administración de procesador/es

La administración del o de los procesadores es una de las más importantes funciones del sistema operativo.

El programa supervisor es quien se encarga de manejar la multiprogramación y el multiprocesamiento, a través de dos componentes denominados planificador, que se encarga básicamente de elegir el programa que se ejecutará en cada oportunidad en que dicha selección sea necesaria; y control de tráfico, cuya función se puede resumir como la del manejo de las interrupciones (iniciación y terminación de entrada/salida, por tiempo, falta de página, finalización de programa, etcétera).

En general, el ordenamiento de asignaciones se organiza alrededor de una especie de lista de aplicaciones, programas o módulos a ser ejecutados y que se encuentran en distintos estados:

- *Ejecutándose:* significa que el programa se encuentra trabajando en ese instante y usando el procesador.
- *Listo para ejecutarse:* significa que el programa se encuentra en condiciones de ser ejecutado, esperando que se le asigne procesador. Para que el supervisor pueda

asignarle procesador a un programa, el estado del mismo debe ser "listo para ejecutarse".

- *Detenido:* significa que por alguna causa se encuentra a la espera de una acción que cambie su estado de detención (por ejemplo: un programa ejecutándose requiere una operación de entrada/salida; el programa pasa a estar en estado "detenido" hasta que el supervisor reciba la señal (otra interrupción) que le indique que ya terminó la operación de entrada/salida. Otro caso sería cuando la página a utilizar no se encuentra en la memoria principal y debe ser traída desde la memoria virtual; mientras se realiza esta tarea, el programa estará en estado "detenido". Al terminar la operación pendiente, en cualquiera de los dos casos, el programa pasa a estado "listo para ejecutarse".

A partir de aquí habrá que especificar en qué ocasiones se consultará la lista para reasignar el procesador. Esto se hace de acuerdo a distintos métodos, de los cuales veremos sólo uno como ejemplo. La asignación del procesador consiste ahora, una vez rastreada la lista, en la elección de uno de los programas en estado "listo para ejecutarse".

Un programa que se encuentre en estado de ejecución podrá ser interrumpido (es decir, que cesará en su procesamiento), por el cambio de estado de "detenido" a "listo para ejecutarse" de otro programa de mayor prioridad o privilegio. Es decir, que el programa en ejecución se puede detener por una causa ajena a su propio trabajo. En este caso, y de acuerdo con los distintos métodos de asignación que se utilicen, podrá forzarse el rastreo de la lista por un cambio de estado en programas de mayor prioridad al que se está ejecutando, produciendo la interrupción del mismo.

Como hemos visto en el capítulo anterior, una interrupción es una comunicación al supervisor del sistema operativo del acaecimiento de un evento que debe ser analizado. Es por ello que frente a una interrupción, cesa la ejecución del programa que se encuentra usando el procesador (en "estado de ejecución") pasándolo al estado "listo para ejecutarse", el supervisor toma el control del procesador para ejecutarse, analizar la interrupción, procesarla y una vez procesada, se continuará ejecutando el programa de la lista (en estado "listo para ejecutarse") que corresponda de acuerdo al método que se utilice.

El que se describe a continuación es alguno de los tantos criterios que se utilizan para asignar el procesador en una mezcla de programas en multiprogramación. El multiprocesamiento es un poco más complejo y excede el objetivo de la explicación.

Un método muy simple y utilizado es el de ciclos de tiempo constante, también denominado en inglés *round robin*, por su mecanismo de ronda. La lista es rastreada a intervalos o ciclos de tiempo constantes. En este caso el ordenamiento de la lista no es significativo, ya que no existen privilegios o prioridades. Comienza a ejecutarse el primero que llega. Empezando con un programa, la lista es rastreada:

- Cuando el programa termina.
- Cuando el programa no puede continuar su ejecución (por ejemplo, por operaciones de entrada/salida).
- Cuando el contador de tiempos genera una interrupción (al cumplirse el ciclo de tiempo).

El rastreo de la lista recomienza en el programa siguiente, volviendo al principio, después del último.

Indudablemente, sólo puede ser seleccionado un programa que esté en estado "listo para ejecutarse" y, en el caso en que todos se encuentren "detenidos", continuará rastreando la lista hasta que una interrupción de fin de operación de entrada/salida (por ejemplo) haga cambiar el estado de un programa a "listo para ejecutarse".

Por otra parte, si un programa está "en ejecución" y se produce una interrupción de fin de operación de entrada/salida de otro programa, luego de procesada la interrup-

ción, se seguirá ejecutando el programa que estaba "en ejecución", ya que no había concluido su ciclo de tiempo. En cambio, si un programa "en ejecución" solicita al sistema operativo una operación de entrada/salida, pasará a estado "detenido", perdiendo el programa la parte del ciclo de tiempo que aún restaba.

Este método no permite que una actividad de mucho tiempo de uso de procesador lo monopolice en algún momento.

c) Administración de procesos a ser ejecutados

A fin de lograr un uso eficiente de los recursos del hardware, una función importante del sistema operativo será la de proveer los servicios necesarios para que se inicie la ejecución de los programas o procesos indicados por el usuario. Esta interfase del sistema operativo con el usuario es denominada *shell* (caparazón) ya que esconde detalles del sistema operativo.

Normalmente el usuario utilizará una interfase gráfica (GUI o *Graphical User Interfase*) y con un simple doble *click* del *mouse*, iniciará la ejecución de un programa. El sistema operativo deberá encargarse de cargarlo en la memoria principal y preparar o asignar los recursos que utilizará la ejecución (contexto de ejecución). Sin embargo, los sistemas operativos también tienen otras alternativas, como mandatos de usuario o sentencias de control de trabajos, que debe primeramente controlar para luego ejecutar lo indicado.

Una vez cargado el o los programas en memoria y asignados los recursos para su ejecución, otras rutinas del supervisor se encargarán de llevar adelante su ejecución, como hemos visto en los puntos anteriores.

d) Administración de dispositivos de entrada/salida

En este punto analizaremos algunos aspectos específicos en la administración de los dispositivos o unidades periféricas de entrada y salida.

d.1) Administración de dispositivos periféricos de entrada/salida

Este aspecto se orienta al uso de canales de entrada/salida y las unidades periféricas. Como ya hemos visto en el capítulo anterior, todas las unidades periféricas se encuentran vinculadas al procesador a través de canales de distinto tipo. Un canal no es simplemente un cable que une la unidad de entrada/salida con la unidad central de procesamiento (con el bus interno), sino que está constituido, además, por una memoria independiente y por un procesador de entrada/salida que puede trabajar simultáneamente con el procesador principal y que permite, en definitiva, facilitar el trabajo en multiprogramación, dejando libre al procesador principal mientras se ejecuta la operación de entrada/salida. Recordemos que, por lo general, las unidades de entrada/salida manejan velocidades muy inferiores al procesador y memoria principal. Si consideramos que cada programa que se está ejecutando puede solicitar distintas operaciones de entrada/salida, vemos que es necesario que otro programa de control superior se encargue de lograr que todas esas operaciones se ejecuten de la manera más eficiente posible.

A su vez, un canal puede realizar de a una operación por vez, o más de una, y puede tener conectadas varias unidades periféricas. Algunos de ellos, exigen que sean del mismo tipo y otros admiten unidades de distinto tipo.

Sin embargo, sabemos que cuando incorporamos alguna unidad muy moderna, el equipo puede no estar en condiciones de manejarla y necesitamos de software del fabricante de la unidad para instalarlo y para que pueda utilizar todas sus características. Es decir, que para el uso de unidades de entrada/salida se requiere el hardware y el software correspondiente. Muchas veces, ese software, manejadores o *drivers* pueden utilizar sin

problemas distintos tipos de unidades del mismo tipo, como por ejemplo los de disco, o pueden manejar la unidad de manera genérica, pero sin poder aprovechar totalmente la funcionalidad de la unidad (como los genéricos de monitor). Los sistemas operativos vienen con gran cantidad de *drivers* de las unidades conocidas hasta que se terminó de desarrollar esa versión del sistema operativo. Y por eso es que muchas veces debemos usar el CD del fabricante (o bajar de internet) para instalar un *driver* no incluido en el sistema operativo y necesario para manejar el dispositivo de manera eficiente y con todas las funcionalidades.

Cada periférico de entrada/salida requiere su propio conjunto especial de instrucciones para cada operación. El sistema operativo y los canales o controladores proporcionan una interfase uniforme que esconde esos detalles de forma que los programas puedan acceder a dichos dispositivos utilizando lecturas y escrituras sencillas, logrando una independencia de la unidades físicas. De esta manera, también, un mismo canal de puerto USB, por ejemplo, puede manejar eficientemente una impresora, un *mouse*, un *pendrive* o un disco externo.

Analicemos cómo maneja el supervisor las operaciones de entrada/salida de una manera simplificada.

Supongamos que un programa solicita una operación de entrada/salida determinada sobre una unidad periférica. Para ello, el programa le entrega al supervisor una serie de datos que le indican el tipo de operación, el soporte o periférico a utilizar, etcétera, efectuando una interrupción, pidiendo la ejecución de la operación de entrada/salida. Lo primero que debe analizar el supervisor es el canal que deberá ser utilizado para atender ese requerimiento. Determinado el canal, deberá verificar el tipo de canal, el estado en que se encuentra y si existen operaciones pendientes para ese canal.

Lo más simple sería que el canal se encontrara libre y que no existieran operaciones pendientes anteriores sobre ese canal. En este caso el supervisor (vía el *driver*) daría al procesador del canal las instrucciones necesarias para que éste efectúe la operación solicitada, y luego analizaría a qué programa le da el control del procesador principal de acuerdo al esquema de multiprogramación que se utilice. Sin embargo, es probable que el canal se encuentre en uso. Si el canal admitiera varias operaciones simultáneamente, puede darse que todos los sub canales estén siendo usados y, en consecuencia, el canal no admita una operación. Si el canal admite una sola operación por vez, al estar en uso, no admitirá tampoco una nueva operación hasta que finalice la que se encuentra realizando.

Cuando el canal efectúe la interrupción indicando la terminación de la operación de entrada/salida que estaba realizando, el supervisor tomará la primera operación de la lista pendiente y, a través del *driver*, le dará al procesador del canal las instrucciones necesarias para cumplirla.

Como vemos, los canales pueden convertirse en un cuello de botella si el equipo no ha sido correctamente configurado.

Sin embargo, todavía no hemos analizado que todos los dispositivos no tienen las mismas características. Por ejemplo, no se puede asemejar el uso de un dispositivo como la unidad de disco al de una impresora.

Podemos decir que existen dispositivos de acceso compartido y otros de acceso dedicado. Veamos un ejemplo: en un paquete de discos puede haber una gran cantidad de archivos. Distintos programas pueden utilizar alternativamente distintos archivos, ya que los brazos móviles permiten el acceso directo. Es decir, que una unidad de discos magnéticos permite que varios programas trabajen con ella alternativamente sin pérdida de eficiencia significativa.

En cambio, si se trata de una cinta magnética, por ser una unidad de acceso secuencial y varios programas utilizan la misma unidad alternativamente, se debería cambiar el soporte para leer o grabar información en el archivo correspondiente. En caso contrario, cada programa leería registros salteados o grabaría en un archivo registros mezclados con los de otro programa.

Lo mismo sucedería con la impresora. Si varios programas la utilizan alternativamente, no se obtendrán dos listados diferenciados, sino uno sólo con informaciones intercaladas de ambos.

Los dispositivos de acceso directo permiten el acceso compartido y alternado entre distintos programas. En cambio, si la unidad es de acceso secuencial, cuando el supervisor la asigna a un programa no iniciará para ese dispositivo operaciones de entrada/salida de otro programa, hasta que el que lo tiene asignado no lo desasigne, ya que se tratará de un dispositivo de acceso dedicado.

d.2) Impresora

La asignación de la impresora puede aparejar grandes inconvenientes de eficiencia en el uso de la computadora, si consideramos trabajar con multiprogramación, es muy probable que más de un programa requiera el uso de la impresora. En este caso, mientras un programa ejecuta, él o los otros deben ser detenidos, hasta que el primero no desasigne la impresora. Además, debemos considerar que mientras se imprime el listado, todos los programas estarán en la memoria, lo que implica un gran desaprovechamiento de este recurso, sobre todo si tenemos en cuenta que la impresora es uno de los dispositivos más lentos de una computadora.

El problema fue considerado suficientemente grave como para justificar el análisis de una solución. Es así como los sistemas operativos cuentan con una facilidad denominada *spooling* (operación periférica simultánea en línea), de utilización optativa.

El sistema es bien sencillo y consiste en que cada vez que un programa asigna la impresora, el sistema operativo genera un archivo en una unidad de disco. Además, cada vez que un programa solicita una operación de impresión, el supervisor desvía la impresión y la graba en el archivo correspondiente. Cada programa no se entera de esta actividad del supervisor que resulta totalmente transparente para ellos. Los distintos listados grabados en disco conforman una cola de impresión que se va imprimiendo realmente, a medida que la impresora va quedando libre.

Esta facilidad permite no sólo que varios programas impriman intercaladamente, sino también que un mismo programa genere más de un listado en la misma ejecución, algo imposible de lograr teniendo una sola impresora.

Es decir, el *spooling* hace independizar totalmente los conceptos de impresora física e impresora lógica. El computador tendrá todas las impresoras virtuales necesarias independientemente de las impresoras físicas.

Debe tenerse en cuenta que esas imágenes de impresión que se grabarán en disco magnético, ocuparán un lugar en dicho soporte, y que debe tenerse presente al configurar el equipo.

Por otro lado, ahora debemos considerar las actividades conducentes a concretar la impresión física del listado, es decir, su pase del soporte de *spool* a impresión. No sólo será necesario que el sistema operativo cumpla estas actividades, sino también otras que hacen al control de esta facilidad. En efecto, entre los requerimientos de esta naturaleza podemos citar los de eliminar un listado; visualizar la lista de los listados pendientes de impresión; especificar la cantidad de copias que se deberán emitir de un listado; especificar que no se destruya el listado luego de la impresión física; cambiar el orden de prioridad para la impresión; determinar la impresión de un listado en particular o a partir de una determinada hoja del mismo; avanzar o retroceder la impresión de un listado en particular o a partir de una determinada hoja del mismo; avanzar o retroceder la impresión del listado de un determinado número de hojas; establecer que un listado no sea impreso hasta que se indique; detener el listado que se está imprimiendo físicamente, conservándolo en el soporte de *spool*; y establecer que si cualquier listado excede una determinada cantidad de hojas, comience a imprimir por segmentos equivalentes a ésta,

establecer en que impresora se imprimirá cada listado, etcétera. Cabe aclarar que cada sistema operativo tendrá más o menos opciones de esta naturaleza. La impresión real del listado podrá realizarse una vez terminada la primer hoja, o bien, cuando se haya completado el listado en su totalidad.

d.3) Discos y unidades lógicamente similares

Como hemos visto anteriormente, los discos magnéticos y las unidades lógicamente similares (como discos de estado sólido, *pendrives*, etcétera) son soportes de acceso directo y compartido, ya que distintos usuarios pueden estar procesando sobre ellos a través de distintos programas y, a su vez, sobre los mismos o diferentes archivos. En consecuencia, las capacidades de almacenamiento que estos soportes proveen, deben ser convenientemente administrados a fin de lograr un uso eficiente.

Históricamente se pueden encontrar distintas formas de administración, cada vez más automáticas y eficientes. Hemos de analizar en este punto la forma en que actualmente se manejan, nuevamente de forma simplificada.

El problema básico de la administración de los espacios de discos está dado por la asignación del espacio a los archivos en los momentos de generación y expansión o reducción debido a su actualización. La mera lectura o modificación de los registros de un archivo no implica problemas de administración del espacio de estos soportes.

El sistema operativo es el encargado de asignar la ubicación del archivo; cada archivo puede dividirse físicamente en unidades de alocación o asignación (vistas en el capítulo anterior), de tal manera que se puedan aprovechar los huecos que van quedando. La asignación es realizada por el sistema operativo tratando siempre de minimizar los huecos y de evitar que las unidades de asignación de un mismo archivo sean distantes físicamente. En el directorio se guardan las direcciones de las distintas unidades de asignación que conforman cada archivo.

Si bien depende de las organizaciones de archivo que veremos más adelante, habitualmente cuando una unidad de asignación se encuentra completa de registros y debe agregarse un nuevo registro, la unidad se subdivide en dos, generándose una nueva unidad del archivo y repartiéndose los registros en las dos unidades y quedando en consecuencia, lugar disponible en ambas para futuras incorporaciones.

Este método permite la extensión de un archivo luego de haber sido creado y resulta particularmente útil para la actualización del contenido de los archivos.

e) Administración de datos

Si bien los programas son los que realizan el manejo lógico de cada archivo que utilizan, no son sus instrucciones las que detallan las actividades necesarias para leer o grabar los datos. Las instrucciones de un programa simplemente establecerán que se deben leer o grabar datos indicando el archivo correspondiente y el área receptora de los datos a ser leídos, o que contiene los que deben ser grabados. Sin embargo, las funciones de manejo de unidades periféricas vistas en el punto anterior, no incluyen el análisis particular referido a cómo buscar o ubicar los datos dentro de un archivo.

Estas funciones son cumplidas por el sistema operativo para determinadas formas de organización de archivos, o bien, por los sistemas de administración de bases de datos. En el Capítulo 10 veremos un poco más de detalle al respecto. Luego, el sistema operativo (los *drivers*) y los canales se encargarán del manejo físico de la unidad de entrada/salida.

Este manejo físico de las unidades periféricas se realiza por sus unidades físicas (bloques de lectura o grabación de acuerdo al soporte de almacenamiento secundario utilizado) y no por las unidades lógicas que procesa el programa (registros lógicos de datos). En un bloque físico puede haber menos de 1 o más de 1 unidad lógica. Nuevamente es el sistema operativo el que se encarga de que el programa pueda trabajar con

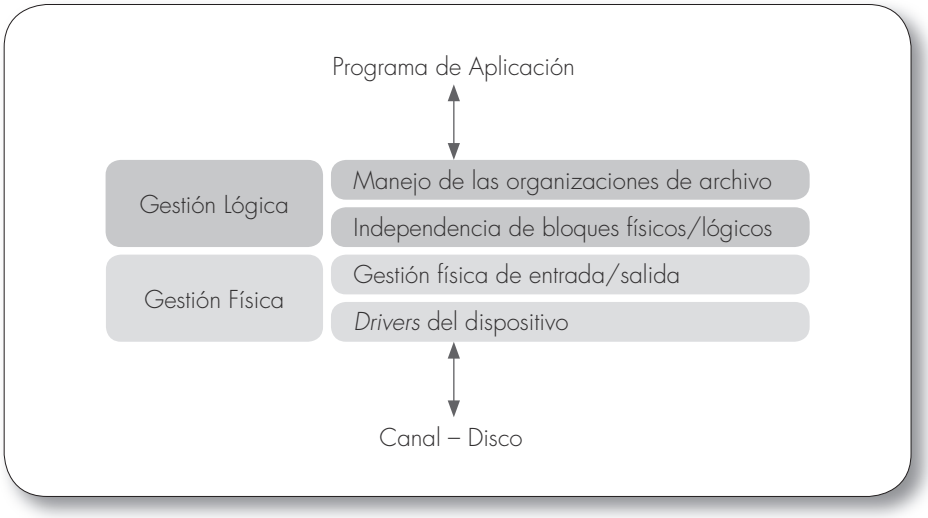
sus unidades lógicas de datos, independizándolo del manejo de las unidades físicas de la unidad de entrada/salida utilizada.

Un programa realizará la misma operación de lectura o grabación, sin importar si utiliza un disco magnético, un disco óptico, un *pendrive* o un disco de estado sólido, a pesar de que cada uno de estos medios tiene características distintas y sus bloques físicos no coinciden.

Resumiendo, el programa se relaciona con la gestión lógica de entrada/salida (que vimos en este punto) que, a su vez, interactúa con la gestión física de la unidad periférica correspondiente (vista en el punto anterior).

En la **Figura 8.4** se representa la arquitectura de software de gestión de archivos.

Figura 8.4
Arquitectura de software de gestión de archivos

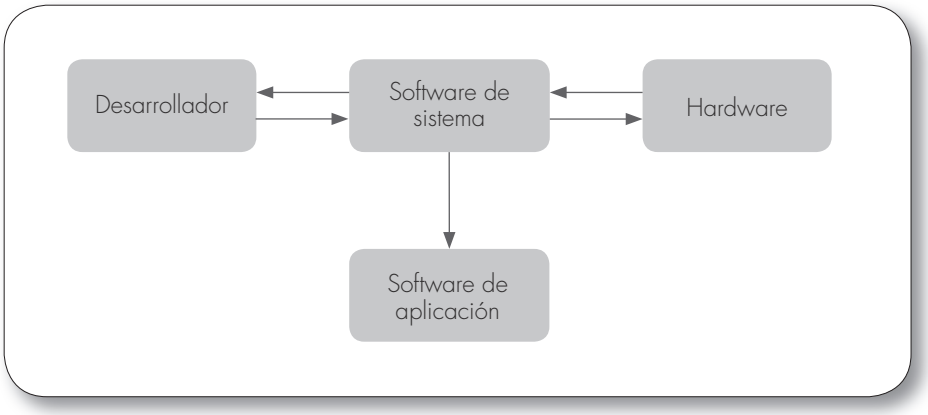


8.5 LENGUAJES DE PROGRAMACIÓN

Como ya hemos visto, para desarrollar un programa hay que usar un lenguaje de programación. Por otro lado, un procesador sólo puede ejecutar instrucciones que estén expresadas en lenguaje de máquina. Cualquier otro lenguaje que utilicemos, necesariamente deberá ser traducido a lenguaje de máquina para poder ser ejecutado. Veremos los traductores en el siguiente punto.

En la **Figura 8.5** vemos cómo el desarrollador, usando software de sistema, genera un software de aplicación.

Figura 8.5
Relación desarrollador-software-hardware



Vamos a analizar la historia de los lenguajes de programación, sus distintos tipos y características.

La primera generación de lenguajes fue el lenguaje de máquina. Cada procesador puede ejecutar instrucciones únicamente en lenguaje de máquina, también llamado código objeto o ejecutable. Si bien no son exactamente lo mismo, consideraremos que código objeto representa un programa ejecutable.

Cada tipo de procesador tiene su propio lenguaje de máquina. Puede haber procesadores compatibles entre sí, aún de distinto fabricante, como es el caso de Intel y AMD, y también procesadores incompatibles, aún del mismo proveedor.

La característica básica de un programa escrito en lenguaje de máquina es que resulta directamente ejecutable por el procesador y, consecuentemente, cada instrucción debe corresponder a una unidad ejecutable y estar expresada en ese lenguaje binario, incluso en las direcciones de memoria. Resulta imaginable la gran dificultad que esto representa, aún para las computadoras de aquella época que trabajaban en mono-programación (sólo un programa en memoria). En breve, resultó más que evidente que era imposible trabajar de esa manera.

En la segunda generación de lenguajes, aparecieron los primeros lenguajes simbólicos, en este caso los lenguajes ensambladores, o también denominados, lenguajes simbólicos de bajo nivel. También cada instrucción escrita representaba una instrucción en lenguaje máquina, pero en lugar de escribirse en binario o hexadecimal, los códigos de instrucción eran nombres mnemotécnicos (por ejemplo, MVC en lugar de 58 para indicar Mover) y los datos se referenciaban con nombres en lugar de sus direcciones de memoria (por ejemplo, SALDO en lugar de 05000B3F). El programa escrito en lenguaje ensamblador se comenzó a denominar código fuente y, como no era ejecutable directamente por el procesador, se requería de un programa traductor (compaginador) que pasara cada instrucción en lenguaje simbólico al equivalente en lenguaje de máquina. Demás está decir el alivio que representó para los programadores de aquella época. Aún así, el nivel de detalle que alcanzaba la programación era importante y sujeta a mayores errores.

Las mejoras de productividad impulsaron el siguiente paso, la tercera generación de lenguajes o lenguajes simbólicos de alto nivel. Aparecieron distintos lenguajes como COBOL (orientado a aplicaciones comerciales), FORTRAN (orientado a aplicaciones matemáticas o ingenieriles), BASIC, RPG, Pascal, PL/1, o C. Estos lenguajes eran más cercanos al lenguaje humano, más fáciles para aprender y sus instrucciones estaban más orientadas al tipo de aplicaciones a desarrollar, y a una mayor facilidad y productividad en la escritura del código fuente. Una instrucción en lenguaje fuente (macroinstrucción) genera normalmente varias instrucciones en lenguaje de máquina. Por sus características y su época, fueron más transportables entre distintos procesadores y sistemas operativos (con pocos cambios, aunque con compiladores diferentes).

Los lenguajes de cuarta generación (4GL) son de aún más alto nivel, con menor cantidad de código fuente para realizar una tarea. Algunos están asociados a sistemas de administración de bases de datos, permitiendo crear una base de datos y las funciones necesarias para la carga de datos y la emisión de informes, de una manera muy sencilla; como, por ejemplo, Access o FoxPro, entre otros. La codificación es menos procedimental y de sintaxis más sencilla para el ser humano. La facilidad de aprendizaje y utilización permite que usuarios no especialistas puedan construir pequeñas aplicaciones usando algunos de estos lenguajes. Si bien las clasificaciones son arbitrarias, podemos incluir, en esta generación, los modernos lenguajes orientados a objetos y visuales, que describimos a continuación.

Existen lenguajes orientados a objetos desde hace varias décadas (por ejemplo, Smalltalk o C++), pero recién en los años noventa su utilización se hizo masiva, con los lenguajes Visual Basic (algunos sostienen que no cumple con todos los aspectos de un

lenguaje orientado a objetos), Visual C++ y Visual C# de Microsoft (cuyos ejecutables corren en plataforma Windows) y el lenguaje multiplataforma Java (para el desarrollo de sitios Web). Su característica principal es que permiten la definición de objetos o clases, que encapsulan datos (atributos o propiedades) y procedimientos, o métodos (que operan sobre los atributos), en código independiente que puede formar parte del ejecutable o ser un ejecutable aparte y, fácilmente reutilizable (hay empresas que desarrollan y comercializan bibliotecas de definiciones de clase). Una vez definida la clase, se puede usar repetidamente y cada una de estas "instancias" "hereda" las características de los atributos y los procedimientos definidos en la clase (los atributos sólo pueden ser accedidos a través de los métodos). Los lenguajes que no están orientados a objetos también manejan datos y procedimientos, pero, aún los lenguajes estructurados de tercera generación, no lo hacen de manera tan independiente y que facilite tanto la reutilización. Las ventajas de los lenguajes orientados a objetos con respecto a lenguajes tradicionales son:

- Profundizan los principios de modularidad (los módulos de código son más independientes al encapsular datos y procedimientos).
- Posibilita una mayor reutilización (al ser más independientes y contar con una biblioteca de objetos probados y posibles de ser reutilizados).
- Mejora la productividad (una mayor reutilización implica menos cantidad de código nuevo a escribir).
- Reduce errores (al reutilizar objetos ya probados y disminuir la cantidad de código nuevo, también se reducen los errores del desarrollo).
- Hace más sencilla su corrección (depuración).
- Facilita el mantenimiento del código fuente.

Los lenguajes de programación visuales, como el Visual Basic (ver **Figura 8.6**), el Visual C++ y el Visual C# de Microsoft o el Delphi de Borland, permiten armar formularios con todos los controles necesarios, sin necesidad de líneas de código. Los formularios típicos de Windows se pueden diseñar seleccionado el tipo de control de una barra de herramientas (nativa y con otros controles adicionales) y "dibujando" la ubicación y tamaño, simplemente usando el *mouse*. Se pueden mover muy fácilmente en el espacio del formulario y asignarle contenido, color, letra y otras características modificando sus propiedades (los controles son clases), sin necesidad de líneas de código. Estos objetos ya tienen procedimientos incluidos y, de una manera muy sencilla, se pueden agregar otros para indicar qué hacer cuando se hace click o doble click con el *mouse*, o cuando el cursor pasa por encima del control, o cuando el control toma o pierde el foco, de una manera muy modular y sencilla, pero usando sentencias del lenguaje de programación. La gran ventaja es que aceleran en gran medida el armado de las interfases gráficas con los usuarios con mucha productividad (al no usar líneas de código que son particularmente extensas y trabajosas para el armado de pantallas), permitiendo ver en forma inmediata el diseño del formulario, reduciendo la posibilidad de errores y facilitando el mantenimiento. En las **Figuras 8.6** y **8.7** se puede ver cómo se diseña un formulario y cómo se ve el mismo formulario en su ejecución.

Por último, también existen un conjunto de lenguajes de programación para el diseño de sitios Web. Desde el muy simple HTML (entre las opciones de un navegador de internet podemos encontrar la de ver el código fuente HTML) hasta PHP o Java, estos lenguajes multiplataforma tienen como objetivo que cualquier navegador de internet pueda ejecutarlos.

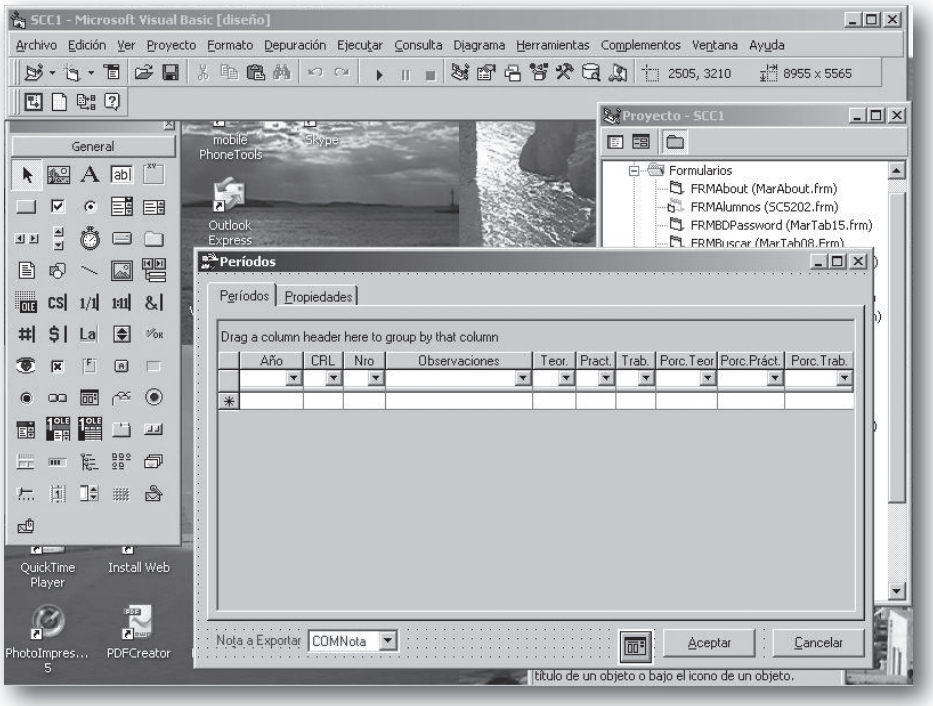


Figura 8.6

Entorno Visual Basic

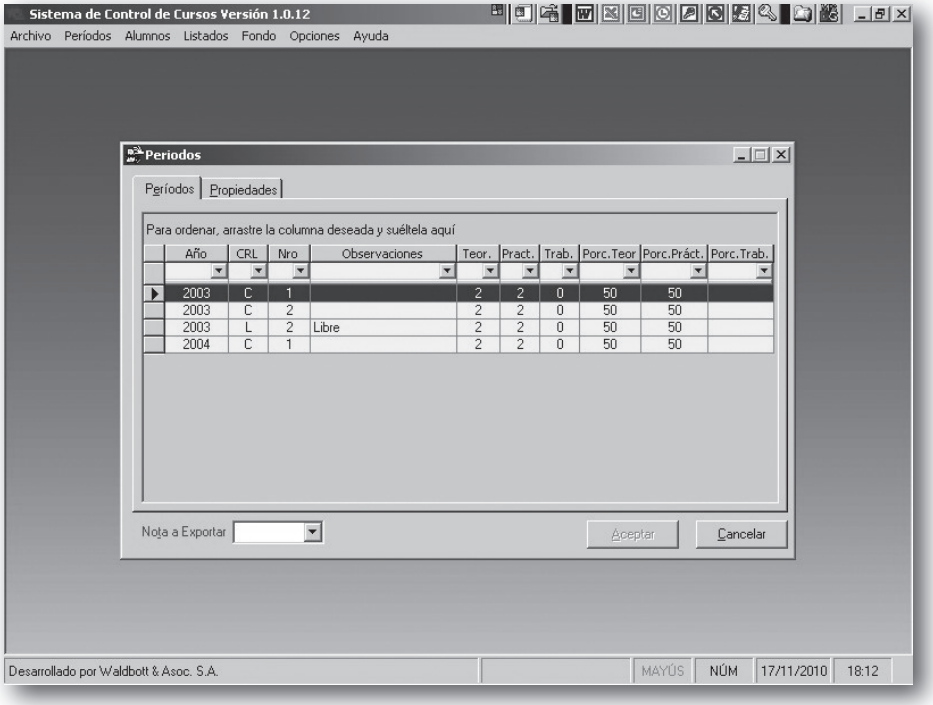


Figura 8.7

Formulario en Ejecución

Roger Pressman en su libro *Ingeniería del Software* presenta una tabla comparativa, donde se presenta la cantidad de líneas de código fuente que llevaría una misma unidad funcional escrita en cada uno de esos lenguajes, por ejemplo, en lenguaje Ensamblador 337, C 162, COBOL 77, C++ 66, Java 63, Visual Basic 47, Access 35, FoxPro y Power-

Builder 32. Como se podrá apreciar, la diferencia es importante y cuanto menos líneas de código, mayor facilidad para aprender y usar el lenguaje, mayor rapidez en el desarrollo (productividad), menos probabilidad de cometer errores y más facilidad para corregirlos y mantener el código en el futuro. Sin embargo, nada es gratis; un mismo programa escrito en un lenguaje de más alto nivel, genera más instrucciones en lenguaje de máquina (más memoria) y consecuentemente es más lento en su ejecución. Esta mayor necesidad de recursos, que fueron requiriendo los lenguajes más complejos, fue acompañada por el abaratamiento y una mayor velocidad y capacidad que, en forma paralela, tuvieron los diferentes componentes de hardware.

Para un sistema operativo o aplicación crítica, se preferirá un lenguaje cuyo ejecutable sea muy eficiente en el uso de los recursos (rápido en su ejecución y que ocupe la menor cantidad de memoria); para una aplicación de uso cotidiano podrá elegirse un lenguaje intermedio, que sea eficiente, pero que sea fácil y rápido para el desarrollo y mantenimiento y para una aplicación ocasional, lo más lógico sería usar un lenguaje que permita un desarrollo muy veloz aunque sea deficiente en velocidad de ejecución o uso de memoria. La importancia de una utilización rápida y oportuna de la tecnología de la información en beneficio de los negocios, los crecientes costos de la hora hombre en comparación con el decreciente costo de memorias y procesadores, han derivado en priorizar la rapidez del desarrollo y la facilidad de mantenimiento por sobre una mayor eficiencia en el uso de los recursos de hardware. Si bien existen muchísimos lenguajes de características diferentes y cada proyecto puede tener funcionalidades propias que orienten hacia un determinado lenguaje, en una organización normalmente se manejan dos o tres lenguajes de programación y se elige entre ellos, ya que, salvo en proyectos muy distintos a los habituales o cuando es un requisito técnico o contractual, el costo de incorporar un nuevo lenguaje supera largamente las ventajas del cambio.

En la **Tabla 8.2** vemos un resumen de las generaciones de lenguajes, sus características, ventajas y el tipo de traductor asociado.

Tabla 8.2 | Lenguajes de programación

Generación	Características	Ventajas	Traductor
1	El programa es directamente codificado en el lenguaje de máquina.	- Muy eficiente en memoria y tiempo de ejecución	- No necesita traductor
2	El programa es codificado en un lenguaje fuente, pero cuyas instrucciones tienen una equivalencia 1 a 1 en instrucciones en lenguaje de máquina.	- Muy eficiente en memoria y tiempo de ejecución - Más fácil para el ser humano	- Compaginador
3	El lenguaje fuente es mucho más cercano al lenguaje humano y cada instrucción genera varias instrucciones en lenguaje de máquina.	- Más fácil para aprender y usar - Más transportable entre sistemas operativos	- Compilador - Intérprete
4	El lenguaje es menos procedimental y de muy alto nivel.	- Más fácil para aprender y usar - Más transportable entre sistemas operativos - Mayor productividad - Más fáciles de mantener	- Compilador - Intérprete - Entorno de desarrollo

Es importante destacar que no hay necesariamente equivalencia entre archivo fuente y archivo objeto o ejecutable. Por distintas razones, en las diferentes generaciones y lenguajes, un archivo programa ejecutable puede originarse en varios archivos de código fuente.

8.6 COMPILADORES, INTÉRPRETES Y ENTORNOS

Los procesos de traducción son muy engorrosos técnicamente y difíciles de explicar, así que veremos sólo sus características conceptuales.

8.6.1 Compaginadores

Con la aparición de los primeros lenguajes simbólicos, los denominados ensambladores o de bajo nivel, también surgieron los primeros traductores de lenguajes simbólicos a lenguaje de máquina, ya que un procesador sólo puede ejecutar directamente, instrucciones expresadas en lenguaje de máquina. Estos primeros traductores se denominaron compaginadores. Por las características de estos lenguajes de bajo nivel, el compaginador realizaba primeramente un control general de sintaxis en todo el programa fuente y si no encontraba errores, generaba la versión ejecutable de ese programa, traduciendo una por una las instrucciones fuente; cada una de ellas se correspondía con una instrucción en lenguaje de máquina. Si había errores, los marcaba en un listado y no generaba el código ejecutable. El proceso de traducción sólo debía volver a realizarse si se modificaba el código fuente; mientras no hubiera modificaciones se reutilizaba el mismo archivo ejecutable que había generado el compaginador y que tenía el programa en lenguaje de máquina ejecutable. Al ejecutar el programa no era necesario el compaginador, dado que el programa fuente ya estaba traducido en lenguaje de máquina, íntegra y previamente.

Dado que estos lenguajes tenían el mismo nivel de detalle del lenguaje de máquina, se agregaron a los compaginadores la posibilidad de "entender" macroinstrucciones que podía definir el programador; utilizando un lenguaje especial, se podía lograr que el compaginador desarrollara estas instrucciones especiales en lenguaje simbólico creadas por un programador, en múltiples instrucciones en lenguaje simbólico nativo y que luego serían a su vez traducidas a lenguaje de máquina.

8.6.2 Compiladores

Los lenguajes simbólicos de alto nivel (sus instrucciones nativas son macroinstrucciones que se traducen en múltiples instrucciones en lenguaje de máquina) utilizaron como traductores, los denominados compiladores o intérpretes. Los compiladores son traductores de características similares a los compaginadores, sólo que asociados a lenguajes de alto nivel.

8.6.3 Intérpretes

Los intérpretes funcionan de manera diferente; la traducción y la ejecución se realizan conjuntamente. Cada instrucción es analizada sintácticamente y, si es correcta, es traducida y ejecutada antes de pasar a la siguiente instrucción. Si se encuentra una instrucción errónea, la ejecución es detenida. La instrucción traducida a lenguaje de máquina no se guarda y, por lo tanto, no se genera un archivo con el programa en lenguaje de máquina y si una instrucción anteriormente traducida y ejecutada debe ser nuevamente ejecutada (como es habitual), será traducida nuevamente. Además, el programa traductor (intérprete) debe estar en memoria todo el tiempo de la ejecución, ya que es el que traduce las instrucciones de lenguaje simbólico de alto nivel a lenguaje de máquina. Consecuen-

temente, una ejecución en intérprete ocupa más memoria principal y es más lenta. Sin embargo, el hecho de disponer del programa fuente en memoria interpretándose, tenía la ventaja de poder ser modificado fácilmente si se encontraban errores de sintaxis o de lógica, además de poder verse fácilmente el contenido de los datos en proceso y de posibilitar la reanudación de la ejecución con los cambios realizados. Unos pocos lenguajes como el BASIC tenían disponibles compiladores e intérpretes.

Por las características señaladas, el intérprete era sumamente ágil y conveniente para la prueba, y puesta a punto del programa por parte del programador y muy desventajoso, y poco seguro, para la ejecución del programa "en régimen" por parte del usuario.

8.6.4 Entornos de desarrollo

Considerando las ventajas de compiladores e intérpretes, surgieron los entornos de desarrollo (ver **Figura 8.6**) que reúnen las características de ambos: permiten interpretar el código fuente para el desarrollo y puesta a punto del programa, y luego generar el archivo correspondiente con el programa en lenguaje de máquina, para su ejecución reiterativa por parte del usuario. Además, brindan una gran cantidad de posibilidades y ayudas al programador para probar, encontrar y corregir errores (*debugging*), así como para llevar el control de versiones del código fuente.

8.7 SOFTWARE PROPIETARIO Y LIBRE

Si bien hay diferentes interpretaciones de software libre y de sus alcances (por su nombre en inglés *freeware*, free puede considerarse "gratuito" o también traducirse como "libre"), a los efectos de nuestro análisis, consideraremos simplemente software gratuito al software que puede ser usado gratuitamente aunque él, o los desarrolladores, no pongan a disposición el código fuente; se considera que si no está disponible el código fuente, no es software libre ya que no es posible estudiar, modificar y adaptar el software, libertades esenciales para considerar libre al software; el software de código abierto (*open source*) será el software con código fuente accesible. Hay software gratuito que no es de código abierto y hay software de código abierto que está disponible también en versiones comerciales pagas. El software pago (en cualquiera de sus formas), sin código fuente accesible, será software propietario.

Para ser precisos vamos a considerar:

- *Software propietario (pago o gratuito):* Cuando el "dueño" del software, al entregarlo, establece restricciones sobre su utilización y/o modificación, al que lo recibe se dice que se trata de un software propietario, o no libre.
  Aun si la pieza de software en cuestión se obtiene gratis, el "propietario" al entregarla puede establecer restricciones con relación a su utilización; por ejemplo, cediendo en forma gratuita el derecho de uso exclusivamente para fines personales, no pudiendo ser utilizada en aplicaciones comerciales, ni copiado, vendido o cedido a terceros. Por ejemplo, hay casos especiales en los cuales un software comercial propietario puede ser usado gratuitamente por los alumnos y docentes de instituciones educativas, para fines pedagógicos, firmando un convenio ad hoc, como es el caso de Microsoft Corporation e IBM, entre otros.
  Cuando un software propietario se obtiene en forma gratuita se dice que es una pieza propietaria gratuita (*freeware*). En algunos casos, con versiones especiales o siempre que el usuario no lo utilice para fines comerciales. Podemos mencionar como ejemplos, el conocido Skype, el antivirus AVG Free, el Acrobat Reader de Adobe (para visualizar archivos PDF), entre otros.

- *Software de código abierto (pago o gratuito):* Cuando hablamos de software de código abierto (*open source*) nos referimos a software cuyo programa fuente es accesible y modificable por el usuario, sin restricciones. Este software puede obtenerse en forma gratuita u onerosa. Puede darse el caso (por otra parte muy común) que se vendan las fuentes del software sin otorgar el derecho a copiarlo y entregarlo a otros (no libre).
- *Software libre:* El concepto de software libre se refiere a aquel cuya licencia de uso (paga o gratuita) garantiza a su receptor la libertad de utilizarlo en lo que quiera, estudiarlo, modificarlo y redistribuirlo, otorgando licencias de igual tipo como desee.

  Una característica distintiva es que una comunidad de desarrolladores voluntarios pueden trabajar para mejorar el software o generar complementos de utilidad. Este trabajo se realiza sin motivaciones económicas (en todo caso un afán de prestigio) y su resultado es accesible por parte de todos los usuarios. Para facilitar el uso del software, se pueden "vender" en versiones comerciales (como Red Hat, por ejemplo).

  La libertad de modificarlo implica la necesidad de que se trate de software de código abierto. Se acepta que esta libertad se condicione en cuanto a la forma de incorporar mejoras y a la obligación de compartir esas mejoras con el resto de la comunidad. Es decir, el receptor puede o no estar obligado (según disponga el proveedor) a entregar libremente todas las mejoras realizadas y compartirlas con el resto de la comunidad. La libertad de redistribuirlo implica que se pueden hacer copias y entregar a terceros, con o sin cargo, independientemente de haberlo obtenido en forma gratuita u onerosa. Es más, un poseedor de licencia puede ofrecer un determinado software sin cargo y otro (el mismo software) en forma onerosa.

  Por lo tanto, una pieza puede ser "libre" sin ser gratuita, pero tiene que ser de código abierto.

Como ejemplos de software de código abierto (código fuente a disposición del usuario) y libre, podemos mencionar el sistema operativo Linux y el Chrome OS, el Symbian para teléfonos celulares, el sistema de administración de base de datos MySQL y el navegador de internet Mozilla Firefox.

El desarrollo de software está amparado por lo derechos de autor. En la Argentina es la Ley 11.723 de Propiedad Intelectual sancionada en 1933, modificada por la Ley 25.036 del 14/10/1998 que incluyó explícitamente el software dentro del marco de los derechos de autor. El registro se realiza en la Cámara de Empresas de Software y Servicios Informáticos (CESSI), que actúa como Ente Cooperador de la Dirección Nacional del Derecho de Autor.

En los casos en que el software es propietario y oneroso (sin código abierto), normalmente no se vende (la compra/venta implica legalmente la transferencia de la propiedad de la cosa), sino que se otorga al usuario el derecho de uso, bajo algún esquema de licenciamiento (licencia de uso). Generalmente, las licencias de uso prohíben la ingeniería inversa, así como duplicar, modificar y transferir el software. El código fuente no estará disponible para los usuarios clientes. Una excepción sería cuando una organización encarga a otra el desarrollo de un software con transferencia final de los derechos de autor y, en ese caso, se configura una compra/venta de software.

Los esquemas de licenciamiento para el uso del software pueden ser sumamente variadas y están más ligadas a la imaginación comercial que a aspectos técnicos. Como ejemplos podemos citar:

- Un valor por cada usuario que utilice el software (puede ser por usuario con el software instalado o por usuarios concurrentes, es decir, la cantidad de usuarios que acceden al uso del software simultáneamente, sin importar cuantos usuarios

tengan instalado el software); muchas veces, el valor de cada usuario no es igual y tiende a decrecer al aumentar la cantidad de usuarios. En software para servidores, el valor puede estar asociado a la cantidad de procesadores del servidor.

- El valor del punto anterior puede ser un valor de única vez, por período de tiempo, de pago mensual (tipo alquiler), de pago anual (frecuentemente denominado canon) o cualquier combinación.
- En ocasiones, el pago puede estar referido a rangos de movimientos permitidos; por ejemplo, hay software de Recursos Humanos o HCM (por su nombre en inglés *Human Capital Management*), cuyo licenciamiento se realiza de acuerdo a rangos de cantidad de empleados.
- Otro caso similar está referido a los software que se utilizan directamente en los servidores del proveedor de software (basados en internet), denominados proveedores de servicios de aplicaciones o ASP (*Application Service Provider*). El valor puede ser por cantidad de transacciones realizadas o también puede ser por suscripción mensual o anual.

Si queremos analizar ventajas y desventajas del software propietario y libre, no podemos dejar de mencionar la importante ventaja de la habitual gratuidad del software libre sobre la habitual onerosidad del software propietario. Si pensamos en un costo por cada usuario, en grandes organizaciones, cada software puede representar un ahorro muy importante en los costos de adquisición.

Si las garantías de un software propietario son relativas de por sí, es presuntamente más difícil que un software libre tenga una corrección de errores ágil, adaptaciones a cambios legales, contables e impositivos, rápidas adaptaciones a nuevos entornos de hardware y software, y nuevas versiones. El giro comercial de un software pago hace más o menos indispensable que se mantenga actualizado y que se generen nuevas versiones. Como en todo, por supuesto que hay ejemplos de lo contrario en unos y en otros.

Por último, una pregunta de múltiples respuestas y argumentos: si usted fuera el gerente de una organización mediana o grande, ¿estaría más seguro con un sistema operativo estándar propietario o con uno libre (con código abierto)? Las dos respuestas tienen sus argumentos.