



## CENTRO DE ESTUDIOS FINANCIEROS

VIRIATO, 52	28010 MADRID	914 44 49 20
PONZANO, 15	28010 MADRID	914 44 49 20
G. DE GRÀCIA, 171	08012 BARCELONA	934 15 09 88
ALBORAYA, 23	46010 VALENCIA	963 61 41 99

[www.cef.es](http://www.cef.es)

[info@cef.es](mailto:info@cef.es)

## Índice Tema 2

1. Introducción.
2. Qué es un sistema operativo.
3. Evolución de los sistemas operativos.
  - 3.1. Historia de los sistemas operativos.
    - 3.1.1. Primera generación (1945-1955): tubos de vacío y tableros enchufables.
    - 3.1.2. Segunda generación (1955-1965): transistores y sistemas de lotes.
    - 3.1.3. Tercera generación (1965-1980): circuitos integrados (CI) y multiprogramación.
    - 3.1.4. Cuarta generación (1980 - hasta hoy): ordenadores personales.
  - 3.2. Variedad de sistemas operativos.
    - 3.2.1. Sistemas operativos de mainframe.
    - 3.2.2. Sistemas operativos de servidor.
    - 3.2.3. Sistemas operativos multiprocesador.
    - 3.2.4. Sistemas operativos de ordenador personal.
    - 3.2.5. Sistemas operativos de tiempo real.
    - 3.2.6. Sistemas operativos integrados.
    - 3.2.7. Sistemas operativos de tarjeta inteligente.
4. Tipos de sistemas operativos.
  - 4.1. Sistemas operativos por su estructura.
    - 4.1.1. Estructura monolítica.
    - 4.1.2. Estructura jerárquica o en capas.
    - 4.1.3. Máquina virtual.
    - 4.1.4. Cliente-servidor (Microkernel).

- 4.2. Sistemas operativos por servicios.
  - 4.2.1. Monousuario.
  - 4.2.2. Multiusuario.
  - 4.2.3. Monotarea.
  - 4.2.4. Multitarea.
  - 4.2.5. Uniproceto.
  - 4.2.6. Multiproceto.
- 4.3. Sistemas operativos por la forma de ofrecer sus servicios.
  - 4.3.1. Sistemas Operativos de Red.
  - 4.3.2. Sistemas Operativos Distribuidos.
- 5. Administración de la memoria.
  - 5.1. Panorama general.
  - 5.2. Manejo de memoria en sistemas monousuario sin intercambio.
  - 5.3. Multiprogramación en memoria real.
    - 5.3.1. El problema de la relocalización.
    - 5.3.2. El problema de la protección.
    - 5.3.3. Particiones fijas o particiones variables.
    - 5.3.4. Los overlays.
  - 5.4. Multiprogramación con memoria virtual.
    - 5.4.1. Paginación pura.
    - 5.4.2. Segmentación pura.
    - 5.4.3. Sistemas combinados.
- 6. Administración de procesos, gestión multitarea.
  - 6.1. Planificación del procesador.
    - 6.1.1. Niveles de planificación.
    - 6.1.2. Objetivos de la planificación.
    - 6.1.3. Características a considerar de los procesos.
    - 6.1.4. Planificación apropiativa o no apropiativa (preemptive or non-preemptive).
    - 6.1.5. Asignación del turno de ejecución.
  - 6.2. Problemas de concurrencia.
- 7. Sistemas de archivos.
  - 7.1. Almacenamiento físico de datos.
    - 7.1.1. Algoritmos de planificación de peticiones.

- 7.1.2. Asignación del espacio de almacenamiento.
  - 7.1.3. Métodos de acceso en los sistemas de archivos.
  - 7.1.4. Operaciones soportadas por el subsistema de archivos.
  - 7.1.5. Algunas facilidades extras de los sistemas de archivos.
- 7.2. Sistemas de archivos aislados.
- 7.3. Sistemas de archivos compartidos o de red.
- 7.4. Tendencias actuales.
- 8. Principios en el manejo de la entrada/salida.
  - 8.1. Dispositivo de entrada/salida.
  - 8.2. Controladores de dispositivos (terminales y discos duros).
  - 8.3. Acceso directo a memoria (DMA).
  - 8.4. Principios en el software de entrada/salida.
    - 8.4.1. Manejadores de interrupciones.
    - 8.4.2. Manejadores de dispositivos.
    - 8.4.3. Software independiente del dispositivo.
    - 8.4.4. Software para usuarios.
  - 8.5. Relojes.
  - 8.6. Plug and Play.
  - 8.7. La interfaz de usuario.
- 9. Vinculación e incrustación de objetos.
  - 9.1. Historia y generalidades.
  - 9.2. ¿Qué es una aplicación OLE?
  - 9.3. Las tecnologías OLE.
  - 9.4. Ventajas de los documentos OLE.





## CENTRO DE ESTUDIOS FINANCIEROS

VIRIATO, 52	28010 MADRID	914 44 49 20
PONZANO, 15	28010 MADRID	914 44 49 20
G. DE GRÀCIA, 171	08012 BARCELONA	934 15 09 88
ALBORAYA, 23	46010 VALENCIA	963 61 41 99

www.cef.es

info@cef.es

### TEMA 2

**Concepto de Sistema Operativo. Componentes y funciones. Características y evolución. Protección de memoria. Gestión multitarea. Sistema de archivos. Vinculación e incrustación de objetos. Plug and Play. Interfaz de usuario.**

#### 1. INTRODUCCIÓN.

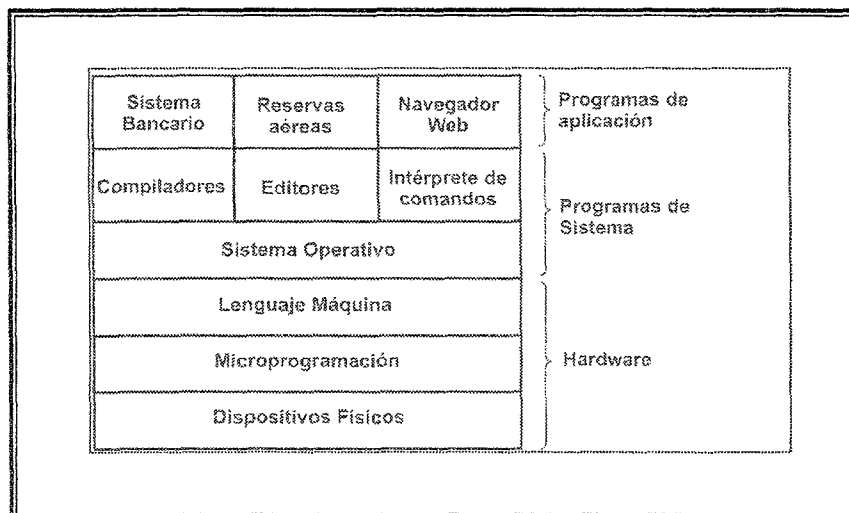
Sin su software, un ordenador es básicamente un montón de hierros sin utilidad. Con su software, un ordenador puede almacenar, procesar y recuperar información, comunicarse con otros ordenadores, compartir la información e interactuar con el entorno a través de dispositivos electromecánicos.

De modo general, el software de un ordenador se puede dividir en dos clases: Los programas de sistema, que manejan la operación del ordenador mismo, y los programas de aplicación, que resuelven problemas de los usuarios.

De todos los programas de sistema, el fundamental de todos es el Sistema Operativo, que controla todos los recursos del ordenador y ofrece la base sobre la cual pueden escribirse y ejecutarse los programas de aplicación.

Un ordenador moderno consta de uno o más procesadores, una memoria principal, relojes, terminales, discos, interfaces de redes, impresoras, otros dispositivos de Entrada/Salida. En conjunto se trata de un sistema complejo. Hacer que todo programador tuviera que involucrarse, en la forma en que trabajan todos los componentes posibles y en todas las combinaciones de ellos que puede tener la más sencilla de las máquinas, habría hecho imposible llegar al estado actual de la informática.

La estrategia que ha evolucionado de manera gradual, para conseguir resguardar a los programadores de la complejidad del hardware, ha consistido en colocar una capa de software, en la parte superior del hardware al descubierto, con el objeto de manejar todas las partes del sistema y presentarse al usuario con una interfaz o máquina extendida, que sea más fácil de entender y programar, ocultando la complejidad interna. Esta capa de software es el Sistema Operativo.



Aunque gran parte del software del sistema es proporcionado por el fabricante del Sistema Operativo, no se considera parte del mismo. El sistema operativo es la parte del software que se ejecuta en modo kernel o modo supervisor.

## 2. QUÉ ES UN SISTEMA OPERATIVO.

Los sistemas operativos responden a dos funcionalidades, básicamente no relacionadas, que responden a diferentes puntos de vista:

Desde el punto de vista del usuario, el sistema operativo tiene como función presentar una máquina virtual o máquina ampliada más fácil de programar y manejar que el hardware asociado. Permite la Creación de programas (edición, compilación, enlazado de librerías), Ejecución de programas (carga, inicialización de dispositivos de E/S, inicialización de ficheros), Acceso a los dispositivos de E/S (instrucciones propias y señales de control de cada dispositivo a través de drivers), Acceso al sistema de ficheros (naturaleza del dispositivo, formato de ficheros, protección frente a múltiples usuarios), Acceso al sistema (sistemas compartidos, gestión de acceso de cada usuario).

Desde el punto de vista de la máquina, el sistema se presenta como un administrador de recursos cuya función es controlar todos los elementos que forman un ordenador, ofreciendo una distribución ordenada y controlada de procesadores, memorias, dispositivos de E/S, etc. entre los diversos programas que compiten por ellos, llevando el control de quién utiliza cuál recurso, concediendo peticiones y resolviendo los conflictos que se generan por el uso concurrente entre los diferentes programas y usuarios.

Un sistema operativo tiene cuatro cometidos principales: la gestión de procesos, la gestión de memoria, el sistema de archivos y la gestión de la Entrada/Salida.

## 3. EVOLUCIÓN DE LOS SISTEMAS OPERATIVOS.

### 3.1. HISTORIA DE LOS SISTEMAS OPERATIVOS.

Los sistemas operativos han ido evolucionando con el paso de los años. Al estar íntimamente ligados con la arquitectura de los ordenadores en los que se ejecutaban, es posible un acercamiento histórico a su evolución a través de las diversas generaciones de ordenadores:

### **3.1.1. Primera generación (1945-1955): tubos de vacío y tableros enchufables.**

Son los primeros sistemas informáticos, los trabajos se lanzan de forma manual sucesivamente. El programador/operador interactúa directamente con el hardware, desde una consola. Los programas se crean en código máquina absoluto y el control de las funciones básicas se realiza mediante paneles enchufables. Se reservan tiempos de uso para los usuarios, durante los cuales son de su exclusivo uso. A partir de 1950 aparecen las tarjetas perforadas que sustituyen la función de los tableros enchufables.

### **3.1.2. Segunda generación (1955-1965): transistores y sistemas de lotes.**

Posteriormente se crean compiladores, ensambladores, bibliotecas de funciones comunes de E/S enlazables con las aplicaciones. Con el tiempo aparecen las unidades de cinta magnética y las impresoras de líneas. La planificación de trabajos (jobs) es manual (lista de reserva). El tiempo de preparación es largo pues se debe cargar el compilador en memoria, el programa fuente, obtener el programa compilado y enlazar las librerías de funciones comunes con el programa objeto. El sistema se organiza formando colas de trabajo (el operador/programador ya no tiene acceso directo a la máquina), gestionadas por un monitor residente siempre en memoria, que se encarga de cargar cada trabajo de la cola y pasarle el control de la UCP para que se ejecute. Una vez ejecutado el programa, éste devuelve el control de la UCP al monitor, que se encarga de cargar el siguiente programa en la cola. El objetivo es minimizar el tiempo de preparación de los trabajos y optimizar así el uso de la UCP. El uso de un monitor residente se apoya en la protección de memoria, la temporización y la ejecución de instrucciones privilegiadas. La preparación de los trabajos se realiza a través de un lenguaje de control de trabajos (JCL).

### **3.1.3. Tercera generación (1965-1980): circuitos integrados (CI) y multiprogramación.**

Aparecen los circuitos integrados que permiten una mayor ventaja precio/rendimiento sobre las máquinas anteriores. La UCP todavía pasa mucho tiempo inactiva debido a las esperas cuando se ejecutan operaciones de E/S. Para evitarlo se cargan varios programas en memoria simultáneamente y se alterna su ejecución. Esto es lo que se denomina multiprogramación. La multiprogramación se apoya en el uso de interrupciones de E/S y de controladores DMA. Para poder desarrollar sistemas multiprogramados, es necesaria la gestión de la memoria y la planificación de trabajos (selección del siguiente programa a ejecutar). Como evolución aparecen los sistemas de tiempo compartido donde el tiempo de procesador se comparte entre programas de varios usuarios pudiendo ser programas interactivos (como procesos de transacciones). A diferencia de la multiprogramación por lotes, donde se pretende maximizar la utilización de la UCP y se utiliza un lenguaje de control de trabajos, el tiempo compartido pretende minimizar el tiempo de respuesta, introduciéndose las órdenes a través de un terminal.

### **3.1.4. Cuarta generación (1980 - hasta hoy): ordenadores personales.**

Aparecen los circuitos LSI (Integración de gran escala), las máquinas se hacen más pequeñas, aumenta la potencia de cálculo y se abaratan los precios hasta permitir que un departamento de una empresa o universidad tenga su propio ordenador. Los sistemas operativos evolucionan hacia sistemas interactivos cuyo objetivo es ser amable con el usuario. Se ponen en marcha las redes de ordenadores personales que ejecutan sistemas operativos en red y sistemas operativos distribuidos.

En un sistema operativo en red, los usuarios tienen conocimiento de la existencia de múltiples ordenadores y pueden acceder a máquinas remotas y copiar archivos de una máquina a la otra. Cada máquina ejecuta su sistema operativo local y tiene un usuario propio. Un sistema operativo distribuido se presenta ante sus usuarios como un sistema uniprocador tradicional, los usuarios no conocen dónde se están ejecutando sus programas o dónde están ubicados sus archivos, toda la complejidad se maneja de forma automática y eficiente por medio del sistema operativo.

Los primeros sistemas operativos de microordenadores personales, obligan a introducir las instrucciones mediante el teclado. Las investigaciones de Doug Engelbart en el Stanford Research Institute, en los años sesenta, le conducen a la invención de la interfaz gráfica de usuario GUI (Graphical User Interface), provista de ventanas, iconos, menús y ratón. Estas ideas fueron adoptadas por los investigadores de Xerox, incorporándolas a sus máquinas.

Steve Jobs, uno de los dos jóvenes que inventaron el ordenador Apple, vio una GUI en una visita al centro de investigación de Xerox, captando el valor potencial de esa tecnología, se dedicó entonces a construir una Apple provista de GUI, que presentara una interfaz amigable para el usuario, lo que significaba que iba dirigida a usuarios que no sólo carecían de conocimientos de ordenadores, sino que no tenían la menor intención de aprender.

Desde entonces, la evolución de los diferentes sistemas operativos de ordenadores personales, ha ido ligada a la existencia en ellos de una interfaz de usuario basada en GUI, que ha mantenido inalterables sus elementos básicos de ventanas, iconos, menús, ratón y ha comenzado a incorporar otros como el reconocimiento de voz, lápices digitales, guantes interactivos, etc.

### **3.2. VARIEDAD DE SISTEMAS OPERATIVOS.**

Toda esta historia y desarrollo han dejado una amplia variedad de sistemas operativos, de los cuales no todos se conocen de forma extensa. Veamos una breve referencia a algunos de ellos.

#### **3.2.1. Sistemas operativos de mainframe.**

En el extremo superior de los ordenadores se hallan los mainframe, ordenadores gigantes que todavía se encuentran en importantes centros de datos corporativos. Se distinguen de los ordenadores personales fundamentalmente por su capacidad de E/S.

Los sistemas operativos de mainframe están orientados al procesamiento de varios trabajos a la vez, con unos elevados requerimientos de E/S. Suelen ofrecer servicios de tres tipos: por lotes, procesamiento de transacciones y tiempo compartido. Un sistema por lotes procesa trabajos rutinarios sin que haya un usuario interactivo presente. El proceso de transacciones maneja solicitudes pequeñas pero en un número muy elevado (miles o decenas de miles por segundo). El tiempo compartido permite a múltiples usuarios ejecutar trabajos en el ordenador, de forma simultánea, dando la impresión de que se dispone del mismo en exclusividad.

Generalmente estos sistemas han sido propietarios y ligados muy íntimamente con la arquitectura hardware que los sustenta. Últimamente, los fabricantes de estos equipos se han abierto al mercado haciendo sus sistemas más estandarizados e incorporando compatibilidad con otros sistemas en sus arquitecturas (LINUX, Windows, UNIX...).

#### **3.2.2. Sistemas operativos de servidor.**

Un nivel por debajo de los sistemas operativos de mainframe, estos sistemas se ejecutan en servidores que son ordenadores personales muy grandes, estaciones de trabajo e incluso mainframes. Dan servicio a múltiples usuarios a través de una red, permitiéndoles compartir recursos hardware y software. Los servidores presentan a los sistemas clientes, servicios disponibles como impresión, archivo o Web. Entre los sistemas operativos típicos están UNIX, Windows2003, y Linux que poco a poco va ganando terreno en el mercado de sistemas operativos para uso profesional.



### **3.2.3. Sistemas operativos multiprocesador.**

Es cada vez más común conseguir aumentar la potencia de proceso, mediante la interconexión de varias UCP en un solo sistema. Dependiendo del modo de hacerlo, hablamos de ordenadores paralelos, multiprocesadores, sistemas en grid, etc. Estas disposiciones necesitan de sistemas operativos especiales, pero con frecuencia éstos son variaciones de los sistemas operativos de servidor, con funciones especiales para comunicación y conectividad.

### **3.2.4. Sistemas operativos de ordenador personal.**

Estos sistemas operativos, los más comúnmente conocidos, se especializan en presentar una buena interfaz a un solo usuario. Se emplean ampliamente para el procesamiento de texto, hojas de cálculo, pequeñas bases de datos, manejo de gráficos sencillos (al conjunto de estas funciones, que se corresponden con el quehacer normal en un entorno de oficina, se le denomina ofimática y a las aplicaciones asociadas paquetes ofimáticos). La familia de sistemas Windows de Microsoft, el sistema operativo Macintosh y Linux son los ejemplos más representativos de sistemas operativos en este segmento.

### **3.2.5. Sistemas operativos de tiempo real.**

Este tipo de sistemas operativos se caracteriza porque su parámetro clave es el tiempo. Empleado principalmente en sistemas de control de procesos industriales, los ordenadores deben capturar datos acerca del proceso de producción y utilizarlos para controlar las máquinas que manejan dentro de unos plazos de tiempo limitados y con frecuencia muy estrictos.

Si una acción debe ejecutarse en cierto momento o intervalo de tiempo obligatoriamente, tendremos un sistema de tiempo real riguroso. Si puede ser aceptable que, de vez en cuando no se cumpla un plazo, tendremos un sistema de tiempo real no riguroso. Los sistemas de audio digital o multimedia pertenecen a esta categoría. VxWorks y QNX son sistemas operativos de tiempo real muy conocidos.

### **3.2.6. Sistemas operativos integrados.**

Considerando sistemas cada vez más pequeños, encontramos los ordenadores de bolsillo (palm-top) y los sistemas integrados. Un ordenador de bolsillo o Asistente Personal Digital, conocido por sus siglas en inglés PDA (Personal Digital Assistant) es un ordenador que coge en el bolsillo de una chaqueta, realizando unas cuantas funciones como libreta de direcciones, bloc de notas, pequeñas hojas de cálculo, etc. Los sistemas integrados son ordenadores que controlan dispositivos que, por lo general, no se consideran ordenadores, como televisores, neveras, teléfonos móviles, etc. Suelen tener características de los sistemas de tiempo real pero con limitaciones de tamaño, memoria y consumo de energía que los hace especiales. Algunos ejemplos de sistemas operativos de ordenadores de bolsillo o integrados son el PalmOS, WindowsCE, Symbian, etc.

### **3.2.7. Sistemas operativos de tarjeta inteligente.**

Los sistemas operativos más pequeños se ejecutan en tarjetas inteligentes, dispositivos usualmente encajados en soportes del tamaño de una tarjeta de crédito, que contienen un chip de CPU y su memoria asociada. Algunos de ellos solamente pueden desempeñar una función y otros realizan varias funciones en la misma tarjeta.

Algunas tarjetas inteligentes están orientadas hacia Java. Esto implica que la ROM de la tarjeta contiene un intérprete de la Máquina Virtual de Java (JVM Java Virtual Machine). Los applets (peque-

ños programas) de Java se descargan a la tarjeta y el intérprete de la JVM los procesa. Algunas tarjetas pueden manejar varios applets al mismo tiempo, dando pie a la multiprogramación y a la necesidad de planificación de tareas.

Debido a las limitaciones de velocidad, capacidad y proceso, suelen ser sistemas operativos muy primitivos.

#### 4. TIPOS DE SISTEMAS OPERATIVOS.

En esta sección se describirán las características que clasifican a los sistemas operativos, básicamente se cubrirán tres clasificaciones: sistemas operativos por su estructura (visión interna), sistemas operativos por los servicios que ofrecen y, finalmente, sistemas operativos por la forma en que ofrecen sus servicios (visión externa).

##### 4.1. SISTEMAS OPERATIVOS POR SU ESTRUCTURA.

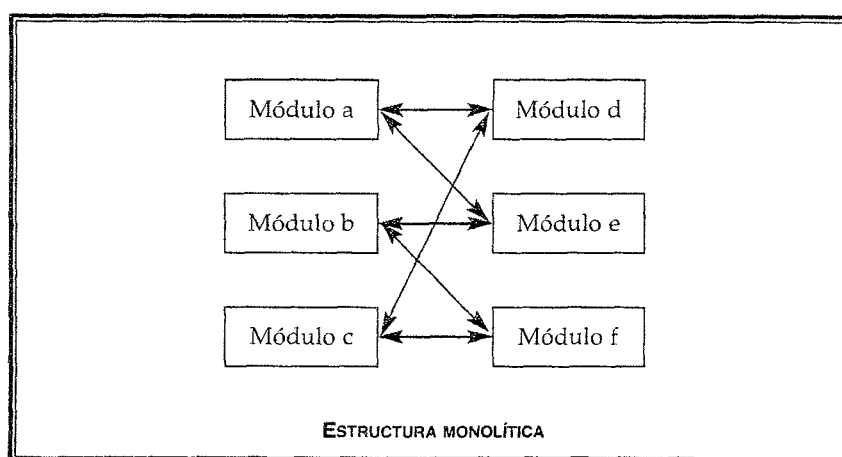
Cuando se construye un sistema operativo, se deben observar dos tipos de requisitos: Requisitos de Usuario: sistema fácil de usar y de aprender, seguro, rápido y adecuado al uso al que se le quiere destinar; Requisitos del Software: donde se engloban aspectos como el mantenimiento, forma de operación, restricciones de uso, eficiencia, tolerancia frente a los errores y flexibilidad.

A continuación se describen las distintas estructuras que presentan los actuales sistemas operativos para satisfacer las necesidades que de ellos se quieren obtener.

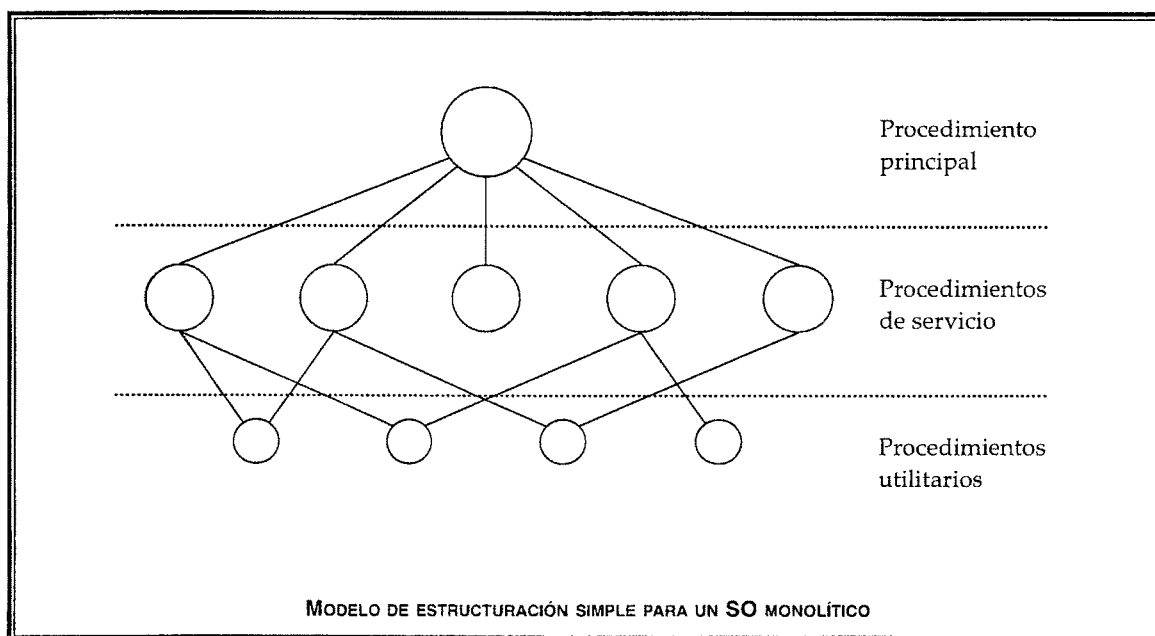
##### 4.1.1. Estructura monolítica.

Es la estructura de los primeros sistemas operativos, constituidos fundamentalmente por un solo programa, compuesto de un conjunto de rutinas entrelazadas de tal forma que, cada una, puede llamar a cualquier otra. Las características fundamentales de este tipo de estructura son: construcción del programa final a base de módulos compilados separadamente que se unen a través del enlazador (linkado); buena definición de parámetros de enlace entre las distintas rutinas existentes, que puede provocar un gran acoplamiento.

Carecen de protecciones y privilegios al entrar a rutinas que manejan diferentes aspectos de los recursos del ordenador, como memoria, disco, etc.



Generalmente están hechos a medida, por lo que son eficientes y rápidos en su ejecución y gestión, pero por lo mismo carecen de flexibilidad para soportar diferentes ambientes de trabajo o tipos de aplicaciones.

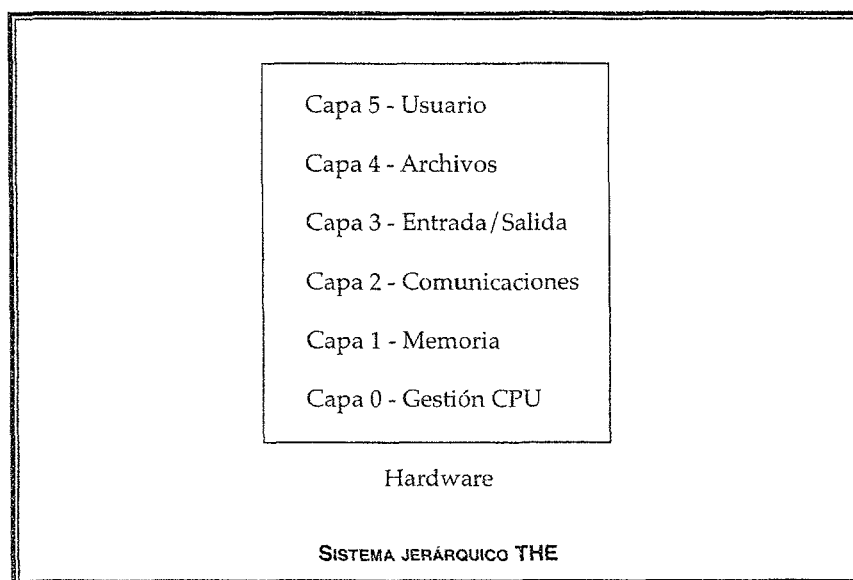


En este tipo de sistemas es posible tener al menos un poco de estructura, derivada del modo en que se llevan a cabo las llamadas al Sistema: los servicios que presta el sistema operativo se solicitan colocando los **parámetros** en un lugar definido (la pila), y ejecutando después una instrucción de llamada (TRAP). La instrucción cambia de modo usuario a modo kernel y transfiere el control al sistema operativo. Entonces, el sistema operativo obtiene los parámetros y determina qué llamada al sistema debe ejecutarse. Esta organización sugiere una estructura básica en donde un procedimiento principal invoca el procedimiento de servicio solicitado, el cual puede hacer uso de uno o varios procedimientos utilitarios que realizan tareas comunes a más de un procedimiento de servicio. Esta división de los procedimientos en tres capas se muestra en la figura anterior.

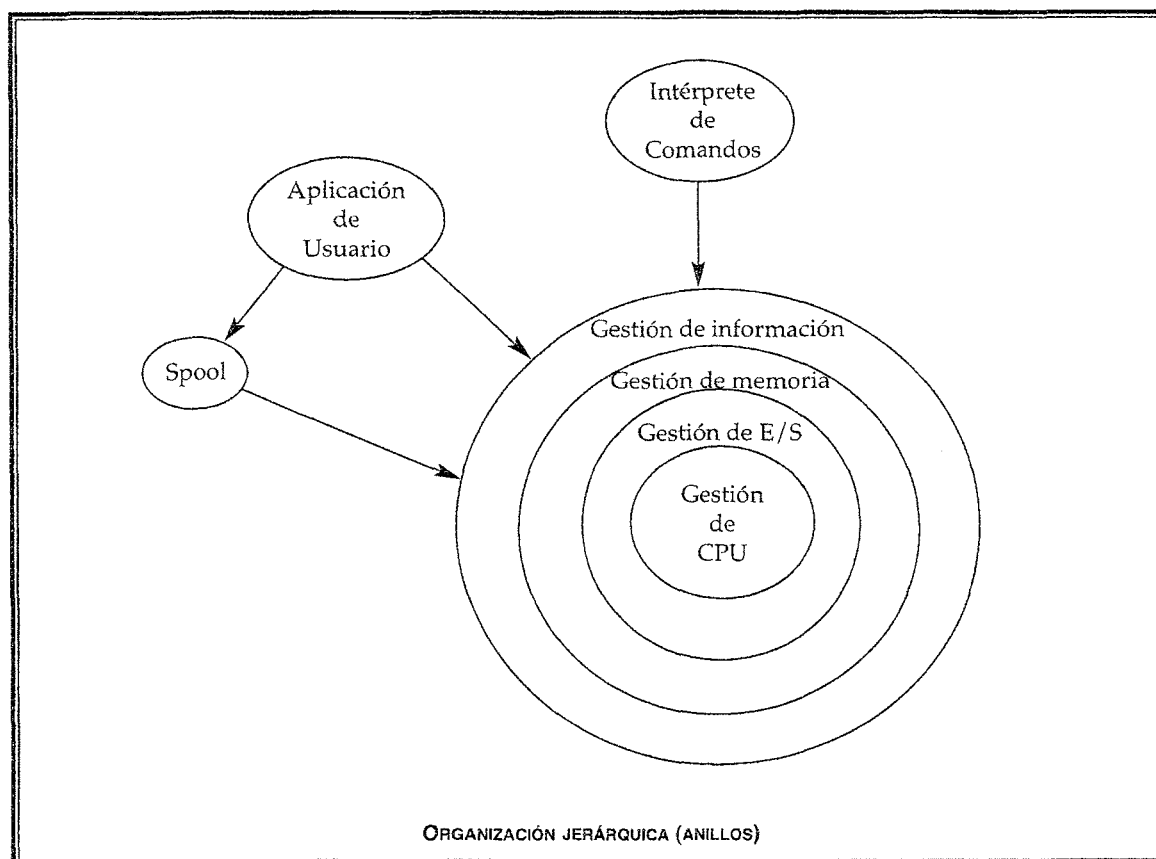
#### 4.1.2. Estructura jerárquica o en capas.

A medida que fueron creciendo las necesidades de los usuarios y se perfeccionaron los sistemas, se hizo necesaria una mayor organización del software del sistema operativo. Se dividió el sistema operativo en **pequeñas partes**, de tal forma que cada una de ellas estuviera perfectamente definida y con un claro **interfaz** con el resto de elementos.

Se constituyó una estructura jerárquica o de niveles en los sistemas operativos, el primero de los cuales fue denominado THE (Technische Hogeschool, Eindhoven), de Dijkstra, que se utilizó con fines didácticos. Se puede pensar también en estos sistemas como si fueran «multicapa». Multics y Unix caen en esa categoría.

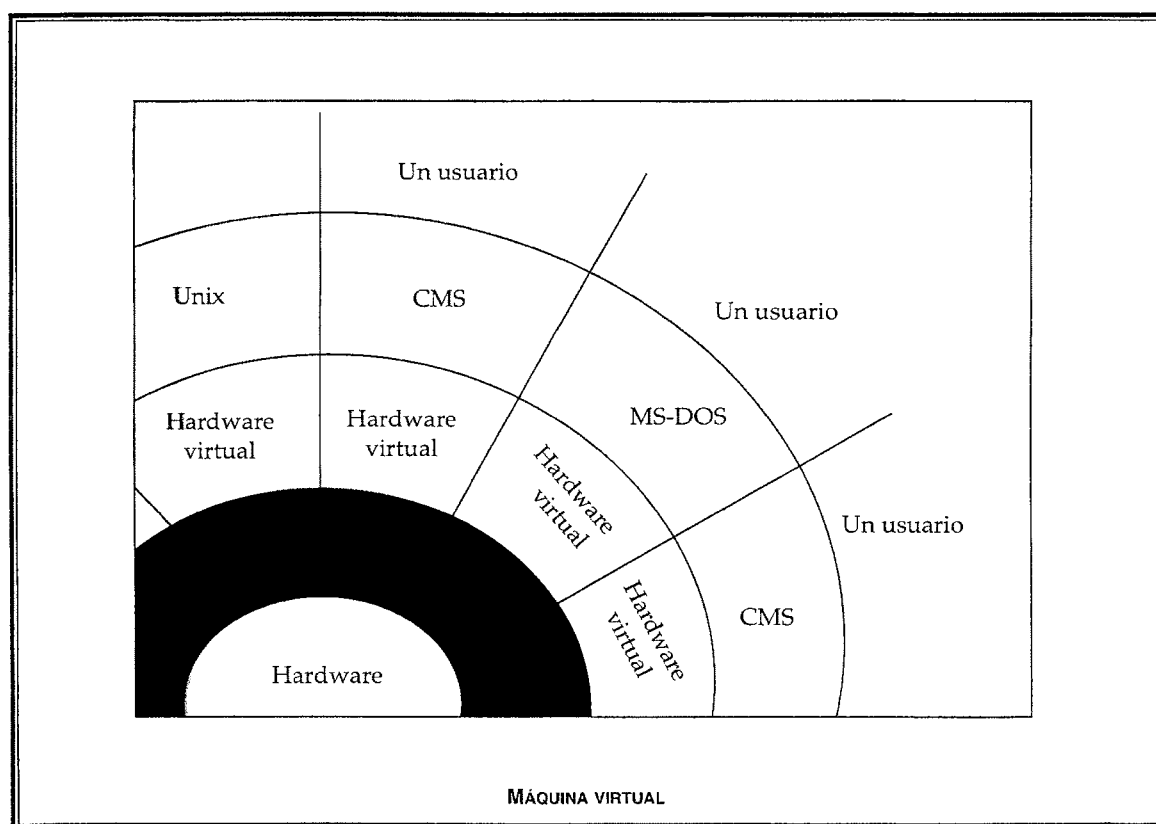


En la estructura anterior se basan prácticamente la mayoría de los sistemas operativos actuales. Otra forma de ver este tipo de sistema es la denominada de anillos concéntricos o «rings». En el sistema de anillos, cada uno tiene una apertura, conocida como puerta o trampa (trap), por donde pueden entrar las llamadas de las capas inferiores. De esta forma, las zonas más internas del sistema operativo o núcleo del sistema estarán más protegidas de accesos indeseados desde las capas más externas. Las capas más internas serán, por tanto, más privilegiadas que las externas.



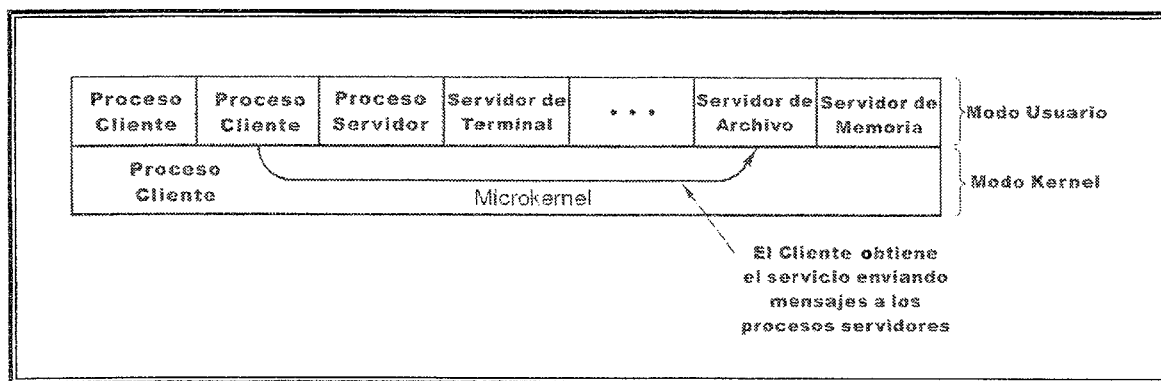
### 4.1.3. Máquina virtual.

Se trata de un tipo de sistemas operativos que presentan una interfaz a cada proceso, mostrando una máquina que parece idéntica a la máquina real subyacente. Estos sistemas operativos separan dos conceptos que suelen estar unidos en el resto de sistemas: la multiprogramación y la máquina extendida. El objetivo de los sistemas operativos de máquina virtual es el de integrar distintos sistemas operativos dando la sensación de ser varias máquinas diferentes. El núcleo de estos sistemas operativos se denomina monitor virtual y tiene como misión llevar a cabo la multiprogramación, presentando a los niveles superiores tantas máquinas virtuales como se soliciten. Estas máquinas virtuales no son máquinas extendidas, sino una réplica de la máquina real, de manera que en cada una de ellas se pueda ejecutar un sistema operativo diferente, que será el que ofrezca la máquina extendida al usuario.



### 4.1.4. Cliente-servidor (Microkernel).

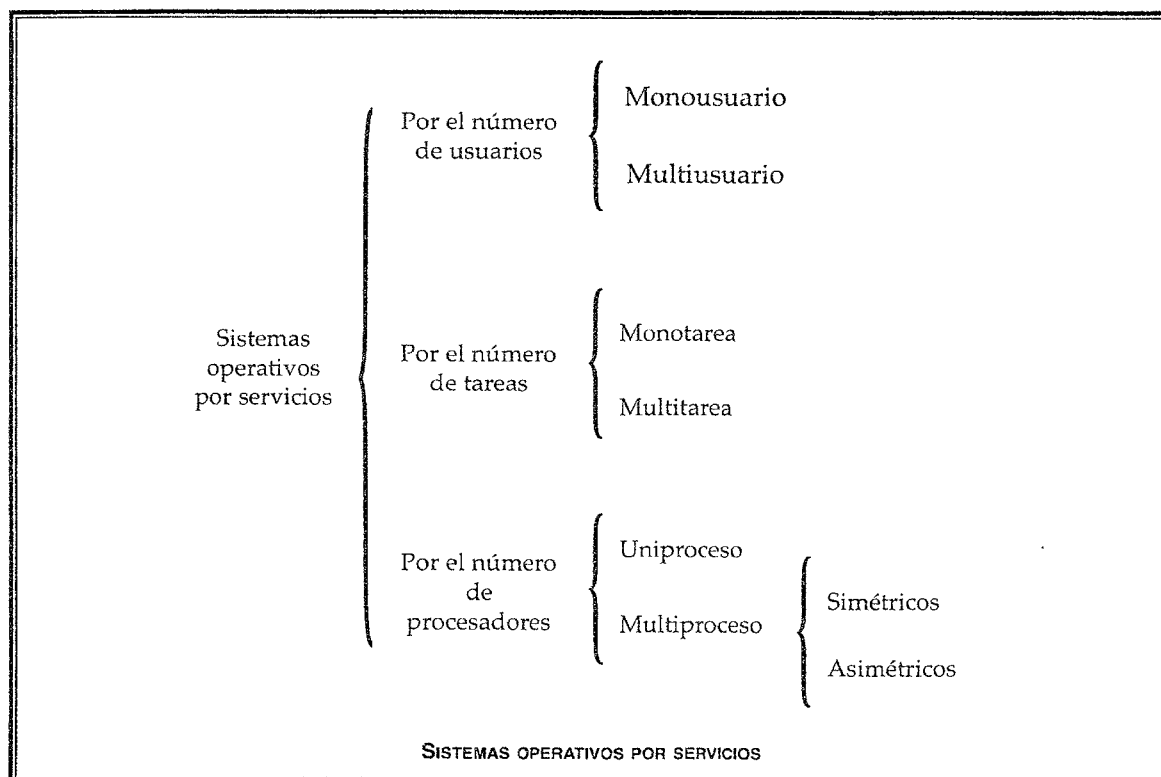
El tipo más reciente de sistemas operativos es el denominado Cliente-servidor, que puede ser ejecutado en la mayoría de los ordenadores, ya sean grandes o pequeños. Este sistema sirve para toda clase de aplicaciones, por tanto, es de propósito general y cumple con las mismas actividades que los sistemas operativos convencionales.



El núcleo tiene como misión establecer la comunicación entre los clientes y los servidores. Los procesos pueden ser tanto servidores como clientes. Por ejemplo, un programa de aplicación normal es un cliente que llama al servidor correspondiente para acceder a un archivo o realizar una operación de entrada/salida sobre un dispositivo concreto. A su vez, un proceso cliente puede actuar como servidor para otro. Este paradigma ofrece gran flexibilidad en cuanto a los servicios posibles en el sistema final, ya que el núcleo provee solamente funciones muy básicas de memoria, entrada/salida, archivos y procesos, dejando a los servidores proveer la mayor parte de los servicios que el usuario final o programador puede usar. Estos servidores deben tener mecanismos de seguridad y protección que, a su vez, serán filtrados por el núcleo que controla el hardware. Actualmente, se está trabajando en una versión de UNIX que contempla en su diseño este paradigma.

#### 4.2. SISTEMAS OPERATIVOS POR SERVICIOS.

Esta clasificación es la más comúnmente usada y conocida desde el punto de vista del usuario final, se comprende fácilmente con el cuadro sinóptico que a continuación se muestra.



#### **4.2.1. Monousuario.**

Los sistemas operativos monousuarios son aquéllos que soportan un usuario a la vez, sin importar el número de procesadores que tenga el ordenador o el número de procesos o tareas que el usuario pueda ejecutar en un mismo instante de tiempo. Los ordenadores personales típicamente se han clasificado en este renglón.

#### **4.2.2. Multiusuario.**

Los sistemas operativos multiusuarios son capaces de dar servicio a más de un usuario a la vez, ya sea por medio de varias terminales conectadas al ordenador o por medio de sesiones remotas en una red de comunicaciones. No importa el número de procesadores en la máquina ni el número de procesos que cada usuario pueda ejecutar simultáneamente.

#### **4.2.3. Monotarea.**

Los sistemas monotarea son aquellos que sólo permiten una tarea a la vez por usuario. Puede darse el caso de un sistema multiusuario y monotarea, en el cual se admiten varios usuarios al mismo tiempo pero cada uno de ellos puede estar haciendo solo una tarea a la vez.

#### **4.2.4. Multitarea.**

Un sistema operativo multitarea es aquél que le permite al usuario estar realizando varias labores al mismo tiempo. Por ejemplo, puede estar editando el código fuente de un programa durante su depuración mientras compila otro programa, a la vez que está recibiendo correo electrónico en un proceso desatendido (en segundo plano o background). Es común encontrar en ellos interfaces gráficas orientadas al uso de menús y el ratón, lo cual permite un rápido intercambio entre las tareas para el usuario, mejorando su productividad.

#### **4.2.5. Uniproceto.**

Un sistema operativo uniproceto es aquél que es capaz de manejar solamente un procesador del ordenador, de manera que si el ordenador tuviese más de uno le sería inútil. El ejemplo más típico de este tipo de sistemas es el DOS y el MacOS.

#### **4.2.6. Multiproceto.**

Un sistema operativo multiproceto se aplica en ordenadores con dos o más procesadores, siendo capaz de usarlos todos para distribuir su carga de trabajo. Generalmente estos sistemas trabajan de dos formas: simétrica o asimétrica. Cuando se trabaja de manera asimétrica, el sistema operativo selecciona a uno de los procesadores como procesador maestro y servirá como pivote para distribuir la carga a los demás procesadores, que reciben el nombre de esclavos. Cuando se trabaja de manera simétrica, los procesos o partes de ellos (hebras o threads) son enviados indistintamente a cualquiera de los procesadores disponibles, teniendo, teóricamente, una mejor distribución y equilibrio en la carga de trabajo bajo este esquema.

Se dice que un thread es la parte activa en memoria y en ejecución de un proceso, el cual puede constar de un área de memoria, un conjunto de registros con valores específicos, la pila y otros valores de contexto. Un aspecto importante a considerar en estos sistemas es la forma de crear aplicaciones para aprovechar los varios procesadores. Existen aplicaciones que fueron hechas para correr en sistemas monoproceso que no toman ninguna ventaja a menos que el sistema operativo o el compilador detecte secciones de código paralelizable, los cuales son ejecutados al mismo tiempo en procesadores diferentes. Por otro lado, el programador puede modificar sus algoritmos y aprovechar por sí mismo esta facilidad, pero esta última opción las más de las veces es costosa en horas hombre y muy tediosa, obligando al programador a ocupar tanto o más tiempo a la paralelización que a elaborar el algoritmo inicial.

#### 4.3. SISTEMAS OPERATIVOS POR LA FORMA DE OFRECER SUS SERVICIOS.

Esta clasificación también se refiere a una visión externa, que en este caso se refiere a la del usuario, el cómo accede a los servicios. Bajo esta clasificación se pueden detectar dos tipos principales: sistemas operativos de red y sistemas operativos distribuidos.

##### 4.3.1. Sistemas Operativos de Red.

Los sistemas operativos de red se definen como aquellos que tienen la capacidad de interactuar con sistemas operativos en otras computadoras a través de un medio de transmisión con el objeto de intercambiar información, transferir archivos, ejecutar comandos remotos y un sin fin de otras actividades. El punto crucial de estos sistemas es que el usuario debe saber la sintaxis de un conjunto de comandos o llamadas al sistema para ejecutar estas operaciones, además de la ubicación de los recursos a los que desee acceder. Por ejemplo, si un usuario en el ordenador hidalgo necesita el archivo matriz.pas que se localiza en el directorio /software/codigo en el ordenador morelos bajo el sistema operativo UNIX, dicho usuario podría copiarlo a través de la red con los comandos siguientes:

```
rcp morelos:/software/codigo/matriz.pas .
```

En este caso, el comando rcp que significa «remotecopy» trae el archivo indicado del ordenador morelos y lo coloca en el directorio donde se ejecutó el mencionado comando. Lo importante es hacer ver que el usuario puede acceder y compartir muchos recursos.

##### 4.3.2. Sistemas Operativos Distribuidos.

Los sistemas operativos distribuidos abarcan los mismos servicios que los sistemas operativos de red, logrando integrar recursos (impresoras, unidades de respaldo, memoria, procesos, unidades centrales de proceso) en una sola máquina virtual a la que el usuario accede de forma transparente. Es decir, ahora el usuario ya no necesita saber la ubicación de los recursos, sino que los conoce por nombre y simplemente los usa como si todos ellos fuesen locales a su lugar de trabajo habitual. Todo lo anterior es el marco teórico de lo que se desearía tener como sistema operativo distribuido, pero en la realidad no se ha conseguido crear uno completamente funcional, por la complejidad que suponen: distribuir los procesos en las varias unidades de procesamiento, reintegrar sub-resultados, resolver problemas de concurrencia y paralelismo, recuperarse de fallas de algunos recursos distribuidos y consolidar la protección y seguridad entre los diferentes componentes del sistema y los usuarios.

Los avances tecnológicos en las redes de área local y la aparición de microprocesadores de 32 y 64 bits han conseguido que, ordenadores más o menos baratos, tengan la suficiente potencia, de forma autónoma, para desafiar en cierto grado a los mainframes. La posibilidad de intercomunicarlos ha su-



gerido la oportunidad de partir procesos de cálculo intensivo, en unidades más pequeñas, distribuyéndolas entre varios microprocesadores, para luego reunir los sub-resultados, creando así una máquina virtual en la red que exceda en poder a un mainframe.

Al sistema integrador de los microprocesadores que presentan la distintas memorias, procesadores, y todos los demás recursos como una sola entidad en forma transparente se le llama sistema operativo distribuido. Las razones para crear o adoptar sistemas distribuidos se justifican principalmente por la necesidad (debido a que los problemas a resolver son inherentemente distribuidos) o porque se desea tener más confiabilidad y disponibilidad de recursos.

En el primer caso tenemos, por ejemplo, el control de los cajeros automáticos en un país. Ahí no es posible ni eficiente mantener un control centralizado, es más, no existe capacidad de cómputo ni de entrada/salida para dar servicio a los millones de operaciones por minuto. En el segundo caso, supóngase que se tienen en una gran empresa varios grupos de trabajo, cada uno necesita almacenar grandes cantidades de información en disco duro con una alta confiabilidad y disponibilidad. La solución puede ser que para cada grupo de trabajo se asigne una partición de disco duro en servidores diferentes, de manera que si uno de los servidores falla, no se deja de prestar el servicio a todos, sino sólo a unos cuantos y, más aún, se podría tener un sistema con discos en espejo (mirror) a través de la red, de manera que si un servidor fallara, el servidor en espejo continuará trabajando sin que el usuario se percate, obteniendo acceso a los recursos de forma transparente.

#### *4.3.2.1. Ventajas de los sistemas distribuidos.*

En general, los sistemas distribuidos (no solamente los sistemas operativos) exhiben algunas ventajas sobre los sistemas centralizados que se describen a continuación.

- **Economía:** el cociente precio/desempeño de la suma del poder de los procesadores separados contra el poder de uno solo centralizado es mejor cuando están distribuidos.
- **Velocidad:** relacionado con el punto anterior, la velocidad sumada es muy superior.
- **Confiabilidad:** si una sola máquina falla, el sistema total sigue funcionando.
- **Crecimiento:** el poder total del sistema puede irse incrementando al añadir pequeños sistemas, lo cual es mucho más difícil en un sistema centralizado y caro.
- **Distribución:** algunas aplicaciones requieren de por sí una distribución física.

Por otro lado, los sistemas distribuidos también exhiben algunas ventajas sobre sistemas aislados. Estas ventajas son:

- **Compartir datos:** un sistema distribuido permite compartir datos más fácilmente que los sistemas aislados, que tendrían que duplicarlos en cada nodo para lograrlo.
- **Compartir dispositivos:** un sistema distribuido permite acceder a dispositivos desde cualquier nodo en forma transparente, lo cual es imposible con los sistemas aislados. El sistema distribuido logra un efecto sinérgico.
- **Comunicaciones:** la comunicación persona a persona es factible en los sistemas distribuidos, en los sistemas aislados no.

- Flexibilidad: la distribución de las cargas de trabajo es factible en los sistemas distribuidos, se puede incrementar el poder de cómputo.

#### 4.3.2.2. Desventajas de los sistemas distribuidos.

Así como los sistemas distribuidos exhiben grandes ventajas, también se pueden identificar algunas desventajas, algunas de ellas tan serias que han frenado la producción comercial de estos sistemas operativos en la actualidad. El problema más importante en la creación de sistemas distribuidos es el software: los problemas de compartición de datos y recursos es tan complejo que los mecanismos de solución generan mucha sobrecarga al sistema haciéndolo ineficiente. El comprobar, por ejemplo, quiénes tienen acceso a algunos recursos y quiénes no. El aplicar los mecanismos de protección y registro de permisos consume demasiados recursos. En general, las soluciones presentes para estos problemas están aún en pañales.

Otros problemas de los sistemas operativos distribuidos surgen debido a la concurrencia y al paralelismo.

Tradicionalmente las aplicaciones se crean para ordenadores de ejecución secuencial, de manera que el identificar secciones de código «paralelizable» es un trabajo arduo, pero necesario para dividir un proceso grande en subprocesos y enviarlos a diferentes unidades de procesamiento para lograr la distribución. Con la concurrencia se deben implantar mecanismos para evitar las condiciones de competencia, las postergaciones indefinidas, el ocupar un recurso y estar esperando otro, las condiciones de espera circulares y, finalmente, los «abrazos mortales» (deadlocks).

Estos problemas de por sí se presentan en los sistemas operativos multiusuario o multitarea, y su tratamiento en los sistemas distribuidos es aún más complejo y, por lo tanto, necesitará de algoritmos más complejos con la inherente sobrecarga esperada.

Por otro lado, en el tema de sistemas distribuidos existen varios conceptos importantes referentes al hardware que no se ven en este trabajo: multicomputadoras, multiprocesadores, sistemas acoplados débil y fuertemente, etc.

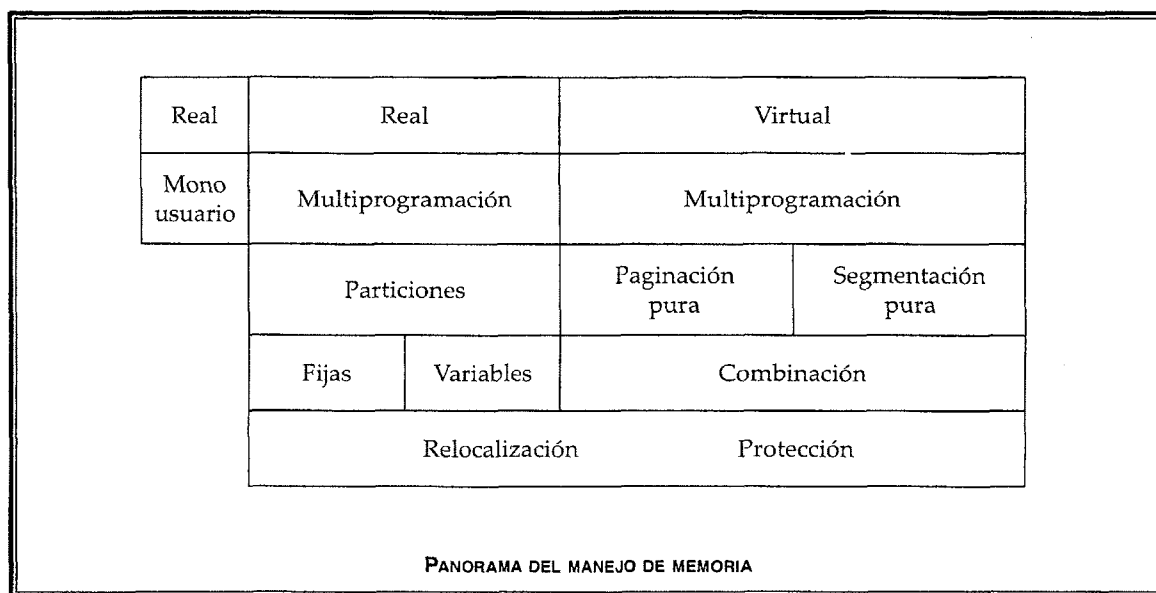
## 5. ADMINISTRACIÓN DE LA MEMORIA.

En esta sección se describirán las técnicas más usuales en el manejo de la memoria, revisando los conceptos relevantes.

Se abarcarán los esquemas de gestión simple de la memoria real, la multiprogramación en memoria real con sus variantes, el concepto de «overlays», la multiprogramación con intercambio y los esquemas de gestión de memoria virtual.

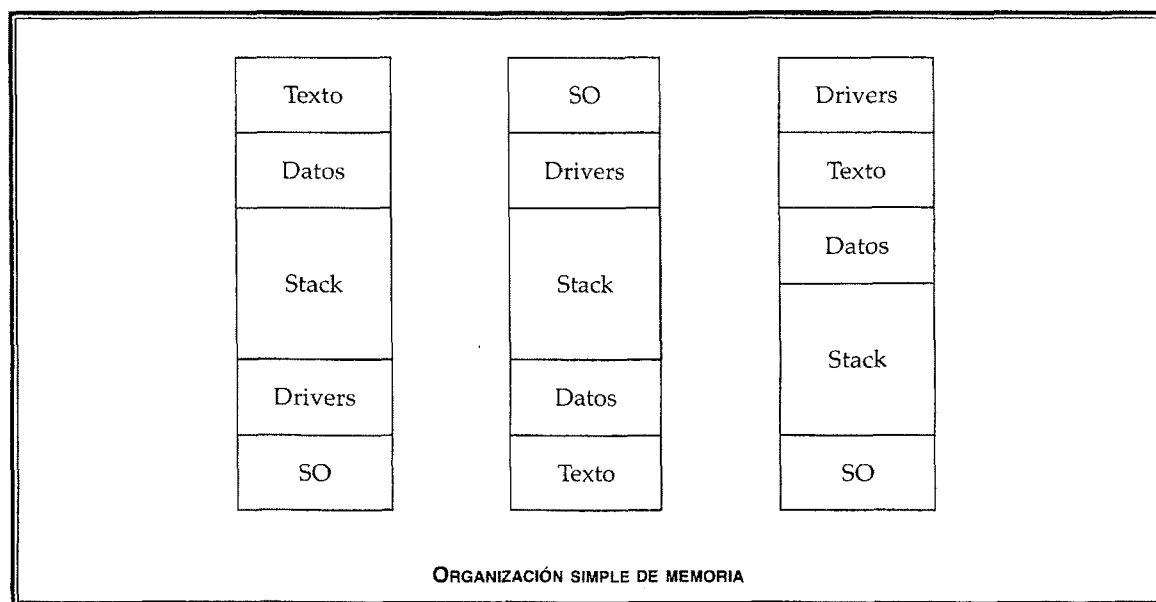
### 5.1. PANORAMA GENERAL.

Un vistazo al material que se va a cubrir en esta sección se muestra en la figura adjunta. Es una gráfica en donde se especifican, en términos generales, los conceptos más importantes en cuanto a las técnicas empleadas en el manejo de memoria.



## 5.2. MANEJO DE MEMORIA EN SISTEMAS MONOUSUARIO SIN INTERCAMBIO.

Este esquema se usa principalmente en sistemas monousuario y monotarea, como son los ordenadores personales con DOS. Bajo este esquema, la memoria real se utiliza para almacenar el programa que se esté ejecutando en un momento dado, con la visible desventaja de estar limitado a la cantidad de RAM disponible. La organización física bajo este esquema es muy simple: el sistema operativo se ubica en las localidades superiores o inferiores de la memoria, seguido por algunos manejadores de dispositivos (drivers). Esto deja un espacio contiguo de memoria disponible que es utilizado por los programas del usuario, dejando generalmente la ubicación de la pila (stack) al último, con el objetivo de que ésta pueda crecer hasta el máximo posible. Estas diferentes opciones se pueden ver en la figura. Como es obvio, bajo estos esquemas no se requieren algoritmos sofisticados para asignar la memoria a los diferentes procesos, ya que éstos son ejecutados secuencialmente conforme van terminando los anteriores.



### 5.3. MULTIPROGRAMACIÓN EN MEMORIA REAL.

En los 60, las empresas e instituciones que habían invertido grandes sumas en la compra de equipo de cálculo se dieron cuenta rápidamente de que los sistemas de proceso por lotes invertían una gran cantidad de tiempo en operaciones de entrada y salida, donde la intervención de la unidad central de proceso era prácticamente nula, y se comenzaron a preguntar cómo hacer que se mantuviera más tiempo ocupada. Fue así como nació el concepto de multiprogramación, el cual consiste en la idea de poner en la memoria física más de un proceso al mismo tiempo, de manera que si el que se está ejecutando en este momento entra en un período de entrada/salida, se puede tomar otro proceso para que haga uso de la unidad central de proceso. De esta forma, la memoria física se dividía en secciones de tamaño suficiente para contener a varios programas.

De esta manera, si un sistema gastaba en promedio 60 por 100 de su tiempo en entrada/salida por proceso, se podía aprovechar más la UCP. Anterior a esto, la UCP se mantenía ese mismo porcentaje ociosa. Con la nueva técnica, el tiempo promedio ocioso disminuye de la siguiente forma: llámese al tiempo promedio que la UCP está ocupada «grado de multiprogramación». Si el sistema tuviese un solo proceso siempre, y éste gastara el 60 por 100 en entrada/salida, el grado de multiprogramación sería  $1 - 60\% = 40\% = 0,4$ . Con dos procesos, para que la UCP esté ociosa se necesita que ambos procesos necesiten estar haciendo entrada/salida, es decir, suponiendo que son independientes, la probabilidad de que ambos estén en entrada/salida es el producto de sus probabilidades, es decir,  $0,6 \times 0,6 = 0,36$ . Ahora, el grado de multiprogramación es  $1 - (\text{probabilidad de que ambos procesos estén haciendo entrada/salida}) = 1 - 0,36 = 0,64$ .

Como se ve, el sistema mejora su uso de UCP en un 24 por 100 al aumentar de uno a dos procesos. Para tres procesos el grado de multiprogramación es  $1 - (0,6)^3 = 0,784$ , es decir, el sistema está ocupado el 78,4 por 100 del tiempo. La fórmula del grado de multiprogramación, aunque es muy idealista, pudo servir de guía para planear un posible crecimiento con la compra de memoria real, es decir, para obtener el punto en que la adición de procesos a RAM ya no incrementaba el uso de la UCP.

Dentro del esquema de multiprogramación en memoria real surgieron dos problemas interesantes: la protección y la relocalización.

#### 5.3.1. El problema de la relocalización.

Este problema no es exclusivo de la multiprogramación en memoria real, se presentó aquí pero se sigue presentando en los esquemas de memoria virtual también. Este problema consiste en que los programas que necesitan cargarse en la memoria real ya están compilados y enlazados, de manera que, internamente, contienen una serie de referencias a direcciones de instrucciones, rutinas y procedimientos que ya no son válidas en el espacio de direcciones de memoria real de la sección en la que se carga el programa. Esto es, cuando se compiló el programa se definieron o resolvieron las direcciones de memoria de acuerdo a la sección de ese momento, pero si el programa se carga en una sección diferente, las direcciones reales ya no coinciden. En este caso, el manejador de memoria puede solucionar el problema de dos maneras: de manera «estática» o de manera «dinámica». La solución «estática» consiste en que todas las direcciones del programa se vuelvan a recalcular al momento en que el programa se carga a memoria, esto es, prácticamente se vuelve a recompilar el programa. La solución «dinámica» consiste en tener un registro que guarde la dirección base de la sección que va a contener al programa. Cada vez que el programa haga una referencia a una dirección de memoria, se le suma el registro base para encontrar la dirección real. Por ejemplo, suponga que el programa se carga en una sección que comienza en la dirección 100. El programa hará referencias a las direcciones 50, 52, 54. Pero el contenido de esas direcciones no es el deseado, sino las direcciones 150, 152 y 154, ya que ahí comienza el pro-

grama. La suma de  $100 + 50$ , etcétera se hace en tiempo de ejecución. La primera solución vale más la pena que la segunda si el programa contiene ciclos y es largo, ya que consumirá menos tiempo en la resolución inicial que la segunda solución en las resoluciones durante la ejecución.

### 5.3.2. El problema de la protección.

Este problema se refiere a que, una vez que un programa ha sido cargado en memoria en algún segmento en particular, nada le impide al programador direccionar (por error o deliberadamente) posiciones de memoria menores que el límite inferior de su programa o superiores a la dirección mayor; es decir, que quiera referenciar localidades fuera de su espacio de direcciones. Obviamente, éste es un problema de protección, ya que no es legal leer o escribir en áreas de otros programas.

La solución a este problema también puede ser el uso de un registro base y un registro límite. El registro base contiene la dirección del comienzo de la sección que contiene al programa, mientras que el registro límite contiene la dirección donde termina. Cada vez que el programa hace una referencia a memoria, se comprueba si ésta pertenece al espacio de direcciones entre su registro base y su registro límite, si no es así se envía un mensaje de error y se aborta el programa.

### 5.3.3. Particiones fijas o particiones variables.

En el esquema de la multiprogramación en memoria real se manejan dos alternativas para asignarle a cada programa su partición correspondiente: particiones de tamaño fijo o particiones de tamaño variable. La alternativa más simple son las particiones fijas. Dichas particiones se crean cuando se enciende el equipo y permanecen con los tamaños iniciales hasta que el equipo se reinicie. Es una alternativa muy vieja, las particiones se realizaban por el operador analizando los tamaños estimados de los trabajos de todo el día. Por ejemplo, si el sistema contaba con 512 KB de RAM, se podía asignar 64 KB para el sistema operativo, una partición más de 64 KB, otra de 128 KB y una mayor de 256 KB.

Este esquema era muy simple, pero inflexible, ya que si surgían trabajos urgentes, por ejemplo, de 400 KB, debían esperar a otro día o reparticionar, inicializando el equipo desde cero. La otra alternativa, que surgió después y como necesidad de mejorar la alternativa anterior, fue crear particiones contiguas de tamaño variable. Para esto, el sistema debía mantener ya una estructura de datos suficiente para saber en dónde había huecos disponibles de RAM y de dónde a dónde había particiones ocupadas por programas en ejecución. Así, cuando un programa requería ser cargado a RAM, el sistema analizaba los huecos para saber si había alguno de tamaño suficiente para el programa entrante, si era así, le asignaba el espacio. Si no, intentaba relocalizar los programas existentes con el propósito de hacer contiguo todo el espacio libre y así obtener un hueco de tamaño suficiente. Si aun así el espacio era insuficiente, el programa se bloqueaba y el sistema optaba por otro. Al proceso mediante el que se juntan los huecos se le denomina de «compactación».

En relación a los esquemas de particiones fijas y particiones variables surgen varios problemas: ¿en base a qué criterio se elige el mejor tamaño de partición para un programa? por ejemplo, si el sistema tiene dos huecos, uno de 18K y otro de 24K para un proceso que desea 15K, ¿cuál se le asigna?, existen varios algoritmos para dar respuesta a la pregunta anterior, los cuales se enumeran y describen a continuación:

- Primer ajuste: se asigna el primer hueco que sea mayor al tamaño deseado.
- Mejor ajuste: se asigna el hueco cuyo tamaño exceda en la menor cantidad al tamaño deseado. Requiere de una búsqueda exhaustiva.

- Peor ajuste: se asigna el hueco cuyo tamaño exceda en la mayor cantidad al tamaño deseado. Requiere también de una búsqueda exhaustiva.
- El siguiente ajuste: es igual que el «primer ajuste» con la diferencia que se deja un apuntador al lugar en donde se asignó el último hueco para realizar la siguiente búsqueda a partir de él.
- Ajuste rápido: se mantienen listas enlazadas separadas de acuerdo a los tamaños de los huecos, para así buscarle a los procesos un hueco más rápido en la cola correspondiente.

Otro problema que se vislumbra desde aquí es que, una vez asignado un hueco, por ejemplo con «el peor ajuste», puede ser que el proceso requiriera 12 KB y que el hueco asignado fuera de 64 KB, por lo cual el proceso va a desperdiciar una gran cantidad de memoria dentro de su partición, este problema se conoce como «fragmentación interna».

Por otro lado, conforme el sistema va avanzando en el día, finalizando procesos y comenzando otros, la memoria se va configurando como una secuencia contigua de huecos y de lugares asignados, provocando que existan huecos, por ejemplo, de 12 K, 28 K y 30 K, que sumados dan 70 K, pero que si en ese momento llega un proceso pidiéndolos, no se le pueden asignar ya que no son localidades contiguas de memoria (a menos que se realice la compactación). Este fenómeno de existencia de huecos no contiguos de memoria se le denomina «fragmentación externa».

De cualquier manera, la multiprogramación fue un avance significativo para el mejor aprovechamiento de la unidad central de proceso y de la memoria misma, así como dio pie para que surgieran los problemas de asignación de memoria, protección y relocalización, entre otros.

#### 5.3.4. Los overlays.

Una vez que surgió la multiprogramación, los usuarios comenzaron a explorar la forma de ejecutar grandes cantidades de código en áreas de memoria muy pequeñas, auxiliados por algunas llamadas al sistema operativo. Es así como nacen los «overlays».

Esta técnica consiste en que el programador divide lógicamente un programa muy grande en secciones que puedan almacenarse en las particiones de RAM. Al final de cada sección del programa (o en otros lugares necesarios) el programador inserta una o varias llamadas al sistema con el fin de descargar la sección presente en RAM y cargar otra, que en ese momento reside en el disco duro u otro medio de almacenamiento secundario. Aunque esta técnica era eficaz (porque resolvía el problema) no era eficiente (ya que no lo resolvía de la mejor manera). Esta solución requería que el programador tuviera un conocimiento muy profundo del ordenador y de las llamadas al sistema operativo. Otra desventaja era la portabilidad de un sistema a otro: las llamadas cambiaban, los tamaños de particiones también. Resumiendo, con esta técnica se podían ejecutar programas más grandes que las particiones de RAM, donde la división del código corría a cuenta del programador y el control a cuenta del sistema operativo.

#### 5.4. MULTIPROGRAMACIÓN CON MEMORIA VIRTUAL.

La necesidad cada vez más imperiosa de ejecutar programas grandes y el crecimiento en potencia de las unidades centrales de proceso empujaron a los diseñadores de los sistemas operativos a implantar un mecanismo para ejecutar automáticamente programas más grandes que la memoria real disponible, esto es, ofrecer «memoria virtual».

La memoria virtual se llama así porque el programador ve una cantidad de memoria mucho mayor que la real. En realidad se trata de la suma de la memoria de almacenamiento primario (memoria real) y una cantidad determinada de almacenamiento secundario (discos). El sistema operativo, en su módulo de manejo de memoria, se encarga de intercambiar programas enteros, segmentos o páginas entre la memoria real y el medio de almacenamiento secundario. Si lo que se intercambia son procesos enteros, se habla entonces de multiprogramación en memoria real, pero si lo que se intercambian son segmentos o páginas, se puede hablar de multiprogramación con memoria virtual.

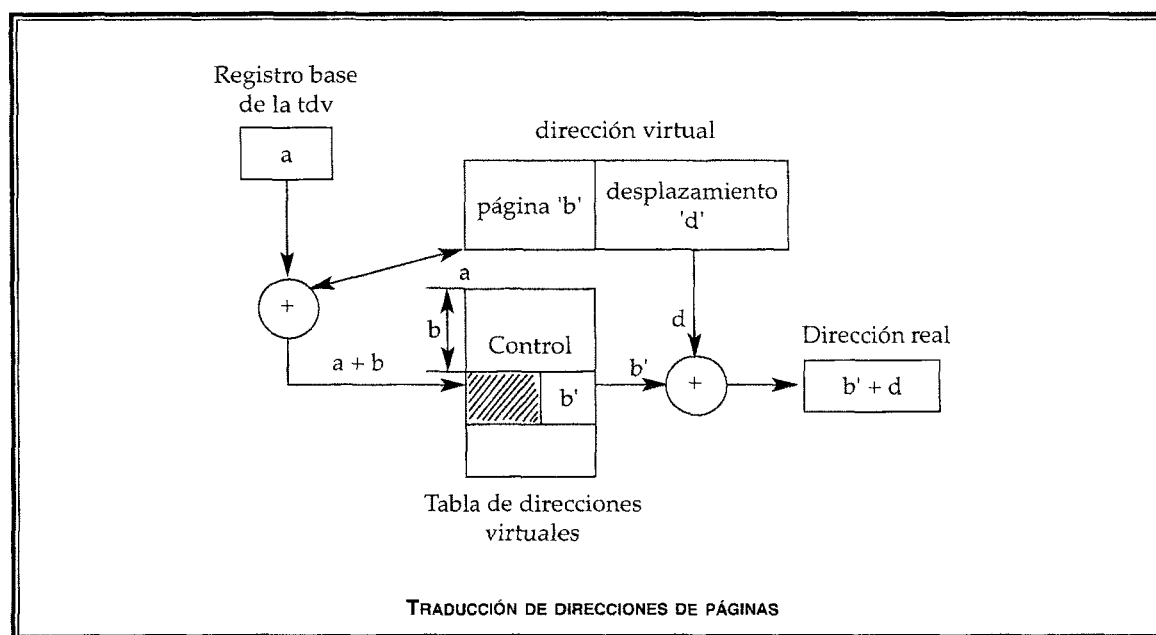
La memoria virtual se apoya en varias técnicas interesantes para lograr su objetivo. Una de las teorías más fuertes es la del «conjunto de trabajo», la cual se refiere a que un programa o proceso no está usando todo su espacio de direcciones en todo momento, sino que existen un conjunto de localidades activas que conforman el «conjunto de trabajo». Si se logra que las páginas o segmentos que contienen al conjunto de trabajo estén siempre en RAM, entonces el programa se ejecutará muy bien.

Otro factor importante es si los programas exhiben un fenómeno llamado «localidad», referido a la tendencia que algunos programas tienen de usar mucho las instrucciones próximas a la instrucción que se está ejecutando actualmente.

#### 5.4.1. Paginación pura.

La paginación pura en el manejo de memoria consiste en que el sistema operativo divide dinámicamente los programas en unidades de tamaño fijo (generalmente múltiplos de 1 kilobyte), los cuales va a intercambiar entre la RAM y el disco. Al proceso de intercambiar páginas, segmentos o programas completos entre RAM y disco se le conoce como «intercambio» o «swapping». En la paginación, se debe cuidar el tamaño de las páginas, ya que si éstas son muy pequeñas el control por parte del sistema operativo para saber cuáles están en RAM y cuáles en disco, sus direcciones reales, etc., crece y provoca mucha «sobrecarga» (overhead). Por otro lado, si las páginas son muy grandes, el overhead disminuye pero entonces puede ocurrir que se desperdicie memoria en procesos pequeños. Debe haber un equilibrio.

Uno de los aspectos más importantes de la paginación, así como de cualquier esquema de memoria virtual, es la forma de traducir una dirección virtual a dirección real. Para explicarlo, obsérvese la figura.



Como se observa, una dirección virtual  $v = (b, d)$  está formada por un número de página virtual  $b'$  y un desplazamiento  $d$ . Por ejemplo, si el sistema ofrece un espacio de direcciones virtuales de 64 kilobytes, con páginas de 4 kilobytes y la RAM sólo es de 32 kilobytes, entonces tenemos 16 páginas virtuales y 8 reales. La tabla de direcciones virtuales contiene 16 entradas, una por cada página virtual. En cada entrada o registro de la tabla de direcciones virtuales se almacenan varios datos: si la página está en disco o en memoria, quién es el dueño de la página, si la página ha sido modificada o es de lectura nada más, etc. Pero el dato que nos interesa ahora es el número de página real que le corresponde a la página virtual. Obviamente, de las 16 virtuales, sólo ocho tendrán un valor de control que dice que la página está cargada en RAM, así como la dirección real de la página, denotada en la figura como  $b'$ . Por ejemplo, supóngase que para la página virtual número 14 la tabla dice que, efectivamente está cargada y es la página real 2 (dirección de memoria 8192). Una vez encontrada la página real, se le suma el desplazamiento, que es la dirección que deseamos dentro de la página buscada ( $b' + d$ ).

La tabla de direcciones virtuales a veces está ubicada en la misma memoria RAM, por lo cual se necesita saber en qué dirección comienza, en este caso, existe un registro con la dirección base denotada por la letra «a» en la figura.

Cuando se está buscando una página cualquiera y ésta no está cargada, surge lo que se llama un «fallo de página» (page fault). Esto es caro para el manejador de memoria, ya que tiene que realizar una serie de pasos extra para poder resolver la dirección deseada y darle su contenido a quien lo pide. Primero, se detecta que la página no está presente y entonces se busca en la tabla la dirección de esta página en disco. Una vez localizada en disco se intenta cargar en alguna página libre de RAM. Si no hay páginas libres se tiene que escoger alguna para enviarla hacia el disco. Una vez escogida y enviada a disco, se marca su valor de control en la tabla de direcciones virtuales para indicar que ya no está en RAM, mientras que la página deseada se carga en RAM y se marca su valor para indicar que ahora ya está en RAM.

Todo este procedimiento es caro, ya que se sabe que los accesos a disco duro son del orden de decenas de veces más lentos que a RAM. En el ejemplo anterior se mencionó que cuando se necesita descargar una página de RAM hacia disco se debe hacer una elección. Para realizar esta elección existen varios algoritmos, los cuales se describen a continuación:

- La primera en entrar, primera en salir (FIFO): se escoge la página que haya entrado primero y esté cargada en RAM. Se necesita que en los valores de control se guarde un dato de tiempo. No es eficiente porque no aprovecha ninguna característica de ningún sistema. Es justa e imparcial.
- La no usada recientemente (NRU): se escoge la página que no haya sido usada (referenciada) en el ciclo anterior. Pretende aprovechar el hecho de la localidad en el conjunto de trabajo.
- La usada menos recientemente (LRU): es parecida a la anterior, pero escoge la página que se usó hace más tiempo, suponiendo que, como lleva mucho sin usarse es muy probable que siga sin usarse en los próximos ciclos. Necesita de una búsqueda exhaustiva.
- La no usada frecuentemente (NFU): este algoritmo toma en cuenta no tanto el tiempo, sino el número de referencias. En este caso cualquier página que se use muy poco, menos veces que alguna otra.
- La menos frecuentemente usada (LFU): es parecida a la anterior, pero aquí se busca en forma exhaustiva aquella página que se ha usado menos que todas las demás.



- En forma aleatoria: elige cualquier página sin aprovechar nada. Es justa e imparcial, pero ineficiente.

Otro dato interesante de la paginación es que ya no se requiere que los programas estén ubicados en zonas de memoria adyacente, ya que las páginas pueden estar ubicadas en cualquier lugar de la memoria RAM.

#### 5.4.2. Segmentación pura.

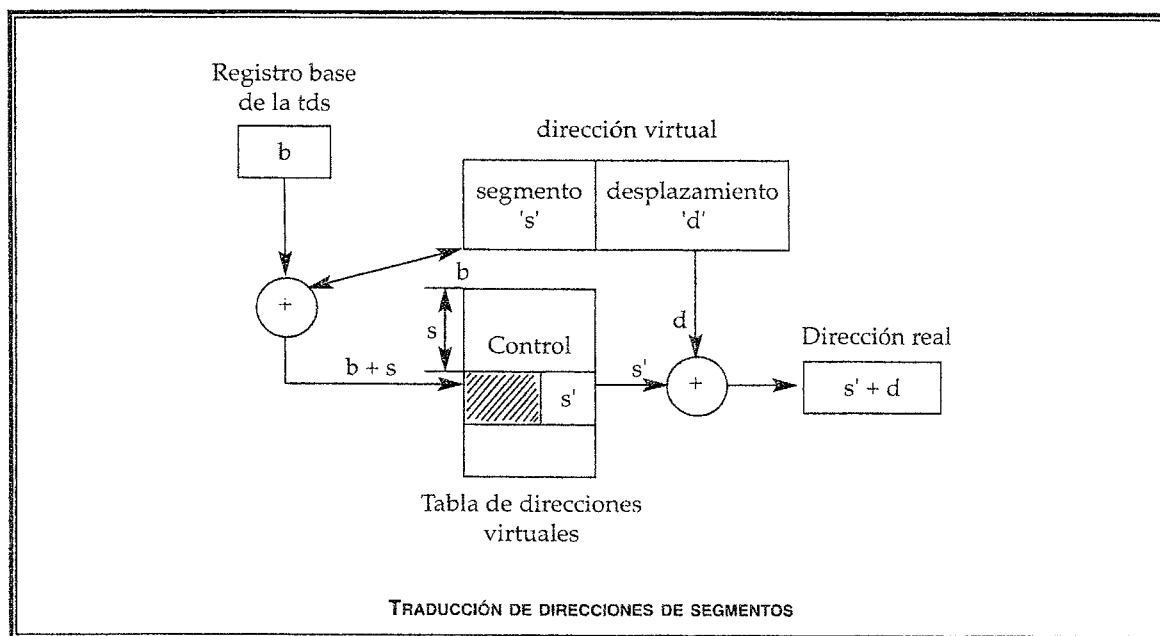
La segmentación aprovecha el hecho de que los programas se dividen en partes lógicas, como son las partes de datos, de código y de pila (stack). La segmentación asigna particiones de memoria a cada segmento de un programa y busca como objetivos el hacer fácil el compartir segmentos (por ejemplo librerías compartidas) y el intercambio entre memoria y los medios de almacenamiento secundario.

Por ejemplo, en la versión de UNIX SunOS 3.5, no existían librerías compartidas para algunas herramientas, por ejemplo, para los editores de texto orientados al ratón y menús. Cada vez que un usuario invocaba a un editor, se tenía que reservar 1 megabyte de memoria. Como los editores son una herramienta muy solicitada y empleada frecuentemente, se dividió en segmentos para la versión 4.x (que a su vez se dividían en páginas), pero lo importante es que la mayor parte del editor es común para todos los usuarios, de manera que la primera vez que cualquier usuario lo invocaba, se reservaba un megabyte de memoria como antes, pero para el segundo, tercero y resto de usuarios, cada editor extra sólo consumía 20 kilobytes de memoria. El ahorro es impresionante. Obsérvese que en la segmentación pura las particiones de memoria son de tamaño variable, en contraste con particiones de tamaño fijo en la paginación pura.

También se puede decir que la segmentación pura tiene una granularidad menor que la paginación por el tamaño de segmentos *versus* tamaño de páginas. Nuevamente, para comprender mejor la segmentación, se debe dar un repaso a la forma en que las direcciones virtuales son traducidas a direcciones reales, con ayuda de las figuras.

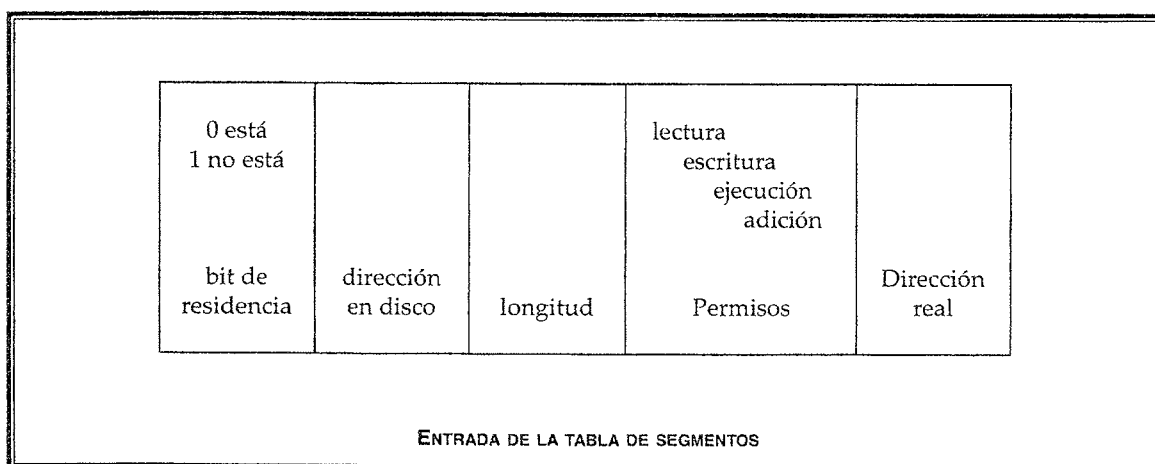
Prácticamente la traducción es igual que la llevada a cabo en la paginación pura, tomando en consideración que el tamaño de los bloques a controlar por la tabla de traducción son variables, por lo cual, cada entrada en dicha tabla debe contener la longitud de cada segmento a controlar. Otra vez se cuenta con un registro base que contiene la dirección del comienzo de la tabla de segmentos. La dirección virtual se compone de un número de segmento (s) y un desplazamiento (d) para ubicar un byte (o palabra) dentro de dicho segmento. Es importante que el desplazamiento no sea mayor que el tamaño del segmento, lo cual se controla simplemente chequeando que ese valor sea mayor que la dirección del inicio del segmento y menor que el inicio sumado al tamaño.

Una vez dada una dirección virtual  $v=(s,d)$ , se realiza la operación  $b + s$  para hallar el registro (o entrada de la tabla de segmentos) que contiene la dirección de inicio del segmento en la memoria real, denotado por  $s'$ . Conociendo la dirección de inicio en memoria real  $s'$  sólo resta encontrar el byte o palabra deseada, lo cual se hace sumándole a  $s'$  el valor del desplazamiento, de modo que la dirección real  $r = s' + d$ .



Cada entrada en la tabla de segmentos tiene un formato similar al mostrado en la figura de ejemplo. Se tienen campos que indican la longitud, los permisos, la presencia o ausencia y dirección de inicio en memoria real del segmento.

Un hecho notable en los sistemas que manejan paginación es que cuando el proceso comienza a ejecutarse se suceden un gran número de fallos de página, porque se referencian muchas direcciones nuevas por primera vez, después el sistema se estabiliza, conforme el número de marcos asignados se acerca al tamaño del conjunto de trabajo.

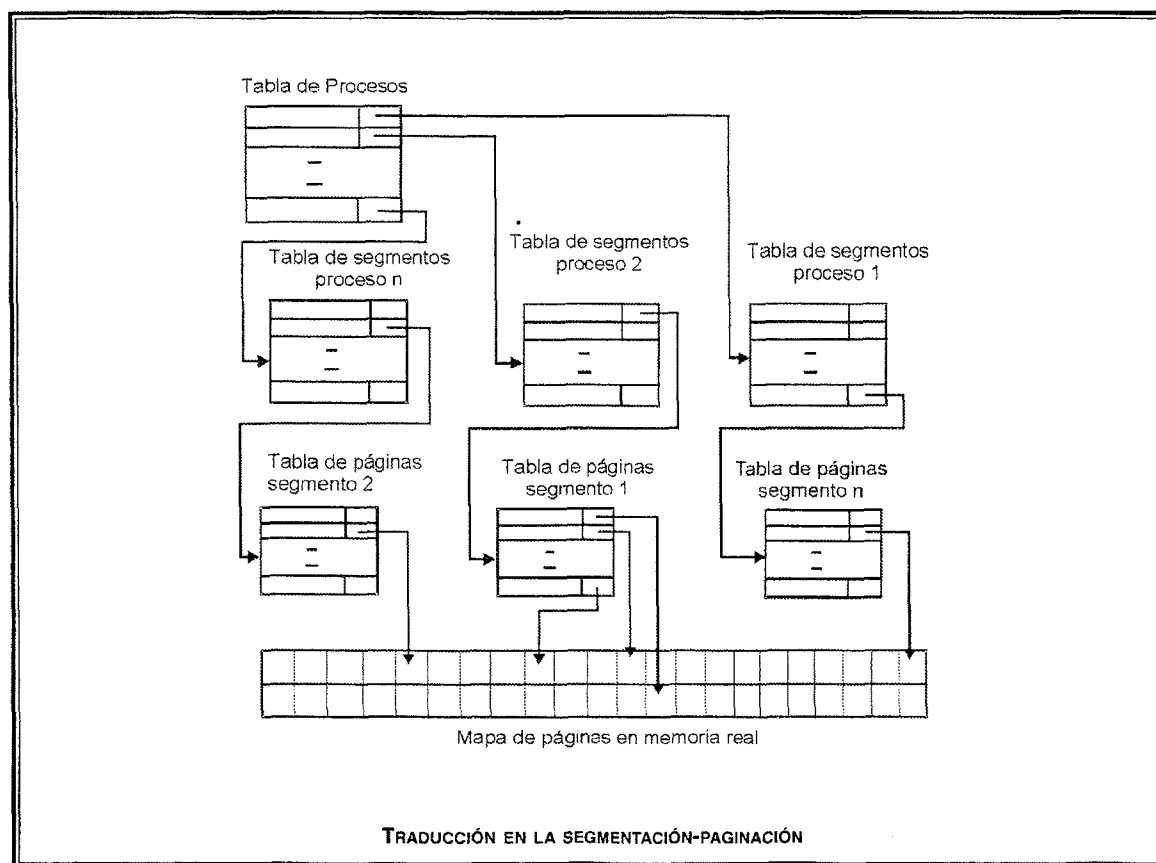


### 5.4.3. Sistemas combinados.

La paginación y la segmentación puras son métodos de manejo de memoria bastante efectivos, aunque la mayoría de los sistemas operativos modernos implantan esquemas combinados, es decir,

combinan la paginación y la segmentación. La idea de combinar estos esquemas se debe a que de esta forma se aprovechan los conceptos de la división lógica de los programas (segmentos) con la granularidad de las páginas. De esta forma, un proceso estará repartido en la memoria real en pequeñas unidades (páginas) cuya liga son los segmentos. También es factible así el compartir segmentos a medida que las partes necesitadas de los mismos se van referenciando (páginas). Para comprender este esquema, nuevamente se verá cómo se traduce una dirección virtual en una posición de memoria real. Para la paginación y segmentación puras se puede decir que el direccionamiento es «bidimensional» porque se necesitan dos valores para hallar la dirección real. Para el caso combinado, se puede decir que se tiene un direccionamiento «tridimensional». En la figura se muestran las partes relevantes para lograr la traducción de direcciones. El sistema debe contar con una tabla de procesos (TP). Por cada renglón de esa tabla se tiene un número de proceso y una dirección a una tabla de segmentos. Es decir, cada proceso tiene una tabla de segmentos. Cuando un proceso hace alguna referencia a memoria, se consulta la TP para encontrar la tabla de segmentos de ese proceso. En cada tabla de segmentos de proceso (TSP) se tienen los números de los segmentos que componen ese proceso. Por cada segmento se tiene una dirección a una tabla de páginas. Cada tabla de páginas tiene las direcciones de las páginas que componen un solo segmento. Por ejemplo, el segmento 'A' puede estar formado por las páginas reales 'a', 'b', 'c', 'p' y 'x'. El segmento 'B' puede estar compuesto de las páginas 'f', 'g', 'j', 'w' y 'z'.

Para traducir una dirección virtual  $v = (s, p, d)$  donde 's' es el segmento, 'p' es la página y 'd' el desplazamiento en la página se hace lo siguiente: primero se ubica de qué proceso es el segmento y se localiza la tabla de segmentos de ese proceso en la TP. Con 's' como índice se encuentra un renglón (registro) en la tabla de segmentos de ese proceso y en ese renglón está la dirección de la tabla de páginas que componen al segmento. Una vez en la tabla de páginas se usa el valor 'p' como índice para encontrar la dirección de la página en memoria real. Una vez en esa dirección de memoria real se encuentra el byte (o palabra) requerido por medio del valor de 'd'.



Ahora, en este esquema pueden generarse dos tipos de fallos: el fallo de página y el fallo de segmento. Cuando se hace referencia a una dirección y el segmento que la contiene no está en RAM (aunque sea parcialmente), se provoca un fallo por falta de segmento y lo que se hace es traerlo del medio de almacenamiento secundario y crearle una tabla de páginas. Una vez cargado el segmento se necesita localizar la página correspondiente, pero ésta no existe en RAM, por lo cual se provoca un fallo de página y se carga de disco y finalmente se puede ya traer la dirección deseada por medio del desplazamiento de la dirección virtual.

La eficiencia de la traducción de direcciones tanto en paginación pura, segmentación pura y esquemas combinados se mejora usando memorias asociativas para las tablas de páginas y segmentos, así como memorias cache para guardar los mapeos más solicitados.

Otro aspecto importante es la estrategia para cargar páginas (o segmentos) a la memoria RAM. Se usan más comúnmente dos estrategias: carga de páginas por demanda y carga de páginas anticipada. La estrategia de carga por demanda consiste en que las páginas solamente son llevadas a RAM si fueron solicitadas, es decir, si se hizo referencia a una dirección que cae dentro de ellas. La carga anticipada consiste en tratar de adivinar qué páginas serán solicitadas en el futuro inmediato y cargarlas de antemano, para que cuando se pidan ya no ocurran fallos de página. Ese «adivinar» puede ser que aproveche el fenómeno de localidad y que las páginas que se cargan por anticipado sean aquellas que contienen direcciones contiguas a la dirección que se acaba de referenciar. De hecho, el sistema operativo VMS usa un esquema combinado para cargar páginas: cuando se hace referencia a una dirección cuya página no está en RAM, se provoca un fallo de página y se carga esa página junto con algunas páginas adyacentes. En este caso, la página solicitada se cargó por demanda y las adyacentes se cargaron por anticipación.

## **6. ADMINISTRACIÓN DE PROCESOS, GESTIÓN MULTITAREA.**

Uno de los módulos más importantes de un sistema operativo es el administrador de procesos y tareas del sistema de cómputo. En esta sección se revisarán dos temas que componen o conciernen a este módulo: la planificación del procesador y los problemas de concurrencia.

### **6.1. PLANIFICACIÓN DEL PROCESADOR.**

La planificación del procesador se refiere a la manera o técnica que se usa para decidir cuánto tiempo de ejecución y cuándo se le asigna a cada proceso del sistema. Obviamente, si el sistema es monousuario y monotarea no hay mucho que decidir, pero en el resto de los sistemas esto es crucial para el buen funcionamiento del sistema.

#### **6.1.1. Niveles de planificación.**

En los sistemas de planificación generalmente se identifican tres niveles: el alto, el medio y el bajo. El nivel alto (scheduler) decide que trabajos (conjunto de procesos) son candidatos a convertirse en procesos compitiendo por los recursos del sistema; el nivel intermedio decide qué procesos se suspenden o reanudan para lograr ciertas metas de rendimiento mientras que el planificador de bajo nivel (dispatcher) es el que decide qué proceso, de los que ya están listos (y que en algún momento pasó por los otros dos planificadores) es al que le toca ahora estar ejecutándose en la unidad central de proceso. En este trabajo se revisarán principalmente los planificadores de bajo nivel porque son los que finalmente eligen al proceso en ejecución.

### 6.1.2. Objetivos de la planificación.

Una estrategia de planificación debe buscar que los procesos obtengan sus turnos de ejecución apropiadamente, conjuntamente con un buen rendimiento y minimización de la sobrecarga (overhead) del planificador mismo. En general, se buscan cinco objetivos principales:

- Justicia o Imparcialidad: todos los procesos son tratados de la misma forma, y en algún momento obtienen su turno de ejecución o intervalos de tiempo de ejecución hasta su terminación exitosa.
- Maximizar la Producción (throughput): el sistema debe de finalizar el mayor número de procesos por unidad de tiempo.
- Minimizar el Tiempo de Respuesta (Turnaround): cada usuario o proceso debe observar que el sistema les responde consistentemente a sus requerimientos.
- Evitar el aplazamiento indefinido: los procesos deben terminar en un plazo finito de tiempo.
- El sistema debe ser predecible: Ante cargas de trabajo ligeras el sistema debe responder rápido y con cargas pesadas debe ir degradándose paulatinamente. Otro punto de vista de esto es que si se ejecuta el mismo proceso bajo cargas similares en todo el sistema, la respuesta en todos los casos debe ser similar.

### 6.1.3. Características a considerar de los procesos.

No todos los ordenadores procesan el mismo tipo de trabajos, y un algoritmo de planificación que en un sistema funciona excelente puede dar un rendimiento pésimo en otro cuyos procesos tienen características diferentes. Estas características pueden ser:

- Cantidad de entrada/salida: existen procesos que realizan una gran cantidad de operaciones de entrada y salida (aplicaciones de bases de datos, por ejemplo).
- Cantidad de uso de UCP: existen procesos que no realizan muchas operaciones de entrada y salida, sino que usan intensivamente la unidad central de procesamiento. Por ejemplo, operaciones con matrices.
- Procesos de lote o interactivos: un proceso de lote es más eficiente en cuanto a la lectura de datos, ya que generalmente lo hace de archivos, mientras que un programa interactivo espera mucho tiempo (no es lo mismo el tiempo de lectura de un archivo que la velocidad en que una persona teclea datos) por las respuestas de los usuarios.
- Procesos en tiempo real: si los procesos deben dar respuesta en tiempo real se requiere que tengan prioridad para los turnos de ejecución.
- Longevidad de los procesos: existen procesos que típicamente requerirán varias horas para finalizar su labor, mientras que existen otros que sólo necesitan algunos segundos.

### 6.1.4. Planificación apropiativa o no apropiativa (preemptive or non-preemptive).

La planificación no apropiativa es aquella en la cual, una vez que a un proceso le toca su turno de ejecución ya no puede ser suspendido, no se le puede arrebatar la unidad central de proceso. Este es-

quema puede ser peligroso, ya que si el proceso contiene accidental o deliberadamente ciclos infinitos, el resto de los procesos pueden quedar aplazados indefinidamente. Una planificación apropiativa es aquella en que existe un reloj que lanza interrupciones periódicas en las cuales el planificador toma el control y se decide si el mismo proceso seguirá ejecutándose o se le da su turno a otro proceso. Este mismo reloj puede servir para lanzar procesos manejados por el reloj del sistema. Por ejemplo, en los sistemas UNIX existen los «cronjobs» y «atjobs», los cuales se programan en base a la hora, minuto, día del mes, día de la semana y día del año.

En una planificación no apropiativa, un trabajo muy grande aplaza mucho a uno pequeño, y si entra un proceso de alta prioridad esté también debe esperar a que termine el proceso actual en ejecución.

### 6.1.5. Asignación del turno de ejecución.

Los algoritmos de la capa baja para asignar el turno de ejecución se describen a continuación:

Por prioridad: los procesos de mayor prioridad se ejecutan primero. Si existen varios procesos de mayor prioridad que otros, pero entre ellos con la misma prioridad, pueden ejecutarse éstos de acuerdo a su orden de llegada o por «round robin». La ventaja de este algoritmo es que es flexible en cuanto a permitir que ciertos procesos se ejecuten primero e, incluso, por más tiempo. Su desventaja es que puede provocar aplazamiento indefinido en los procesos de baja prioridad. Por ejemplo, suponga que existen procesos de prioridad 20 y procesos de prioridad 10. Si durante todo el día terminan procesos de prioridad 20 al mismo ritmo que entran otros con esa prioridad, el efecto es que los de prioridad 10 estarán esperando por siempre. También provoca que el sistema sea impredecible para los procesos de baja prioridad.

El trabajo más corto primero (SJF Shortest Job First): es difícil de llevar a cabo porque se requiere saber o tener una estimación de cuánto tiempo necesita el proceso para terminar, pero si se sabe, se ejecutan primero aquellos trabajos que necesitan menos tiempo y de esta manera se obtiene el mejor tiempo de respuesta promedio para todos los procesos. Por ejemplo, si llegan 5 procesos A, B, C, D y E cuyos tiempos de UCP son 26, 18, 24, 12 y 4 unidades de tiempo, se observa que el orden de ejecución será E, D, B, C y A (4, 12, 18, 24 y 26 unidades de tiempo respectivamente). En la tabla siguiente se muestra en qué unidad de tiempo comienza a ejecutarse cada proceso y como todos comenzaron a esperar desde la unidad cero, se obtiene el tiempo promedio de espera.

PROCESO	ESPERA DESDE	TERMINA	TIEMPO DE ESPERA
A	0	4	4
B	0	16	16
C	0	34	34
D	0	58	58
E	0	84	84
Tiempo promedio = $(4 + 16 + 34 + 58 + 84)/5 = 39$ unidades			

El primero en llegar, primero en ejecutarse: es muy simple, los procesos reciben su turno conforme llegan.

La ventaja de este algoritmo es que es justo y no provoca aplazamiento indefinido. La desventaja es que no aprovecha ninguna característica de los procesos y puede no servir para un proceso de tiempo real. Por ejemplo, el tiempo promedio de respuesta puede ser muy malo comparado con el logrado por el del trabajo más corto primero. Retomando el mismo ejemplo que en el algoritmo anterior, obtenemos un tiempo de respuesta promedio  $(26 + 44 + 68 + + 80 + + 84)/5 = 60$  unidades, el cual es muy superior a las 39 unidades que es el mejor tiempo posible.

Round Robin: también llamado por turno, consiste en darle a cada proceso un intervalo de tiempo de ejecución (llamado time-slice), y cada vez que se vence ese intervalo se copia el contexto del proceso a un lugar seguro y se le da su turno a otro proceso. Los procesos están ordenados en una cola circular. Por ejemplo, si existen tres procesos, el A, B y C, dos pasadas del planificador darían sus turnos a los procesos en el orden A, B, C, A, B, C. La ventaja de este algoritmo es su simplicidad, es justo y no provoca aplazamiento indefinido.

El tiempo restante más corto: es parecido al del trabajo más corto primero, pero aquí se está calculando en todo momento cuánto tiempo le resta para terminar a todos los procesos, incluyendo los nuevos, y aquel que le quede menos tiempo para finalizar es escogido para ejecutarse. La ventaja es que es muy útil para sistemas de tiempo compartido porque se acerca mucho al mejor tiempo de respuesta, además de responder dinámicamente a la longevidad de los procesos; su desventaja es que provoca más sobrecarga porque el algoritmo es más complejo.

La tasa de respuesta más alta: este algoritmo concede el turno de ejecución al proceso que produzca el mayor valor de la siguiente formula:

$$\text{valor} = (\text{tiempo que ha esperado} + \text{tiempo total para terminar}) / \text{tiempo total para terminar.}$$

Es decir, dinámicamente el valor se va modificando y mejora un poco las deficiencias del algoritmo del trabajo más corto primero.

Por política: una forma de asignar el turno de ejecución es por política, en la cual se establece algún reglamento específico que el planificador debe obedecer. Por ejemplo, una política podría ser que todos los procesos reciban el mismo tiempo de uso de UCP en cualquier momento. Esto significa, por ejemplo, que si existen 2 procesos y han recibido 20 unidades de tiempo cada uno (tiempo acumulado en time-slices de 5 unidades) y en este momento entra un tercer proceso, el planificador le dará inmediatamente el turno de ejecución por 20 unidades de tiempo. Una vez que todos los procesos están «parejos» en uso de UCP, se les aplica «round robin».

## 6.2. PROBLEMAS DE CONCURRENCIA.

En los sistemas de tiempo compartido (aquellos con varios usuarios, procesos, tareas, trabajos que reparten el uso de UCP entre éstos) se presentan muchos problemas debido a que los procesos compiten por los recursos del sistema.

Imagine que un proceso está escribiendo en la unidad de cinta y se le termina su turno de ejecución e inmediatamente después el proceso elegido para ejecutarse comienza a escribir sobre la misma cinta. El resultado es una cinta cuyo contenido es un desastre de datos mezclados. Así como la cinta, existen una multitud de recursos cuyo acceso debe ser controlado para evitar los problemas de la concurrencia.

El sistema operativo debe ofrecer mecanismos para sincronizar la ejecución de procesos: semáforos, envío de mensajes, «pipes», etc. Los semáforos son rutinas de software (que en su nivel más interno se auxilian del hardware) para lograr exclusión mutua en el uso de recursos. Para entender este y otros mecanismos es importante entender los problemas generales de concurrencia, los cuales se describen a continuación:

- Condiciones de carrera o competencia: la condición de carrera (race condition) ocurre cuando dos o más procesos acceden a un recurso compartido sin control, de manera que el resultado combinado de este acceso depende del orden de llegada. Suponga, por ejemplo, que dos clientes de un banco realizan cada uno una operación en cajeros diferentes al mismo tiempo.

El usuario A quiere hacer un depósito. El B una extracción. El usuario A comienza la transacción y lee su saldo que es 1.000. En ese momento pierde su turno de ejecución (y su saldo queda como 1.000) y el usuario B inicia la extracción: lee el saldo que es 1.000, retira 200 y almacena el nuevo saldo que es 800 y termina. El turno de ejecución regresa al usuario A el cual hace su depósito de 100, quedando saldo = saldo + 100 = 1.000 + 100 = 1.100. Como se ve, la extracción se perdió y eso le encanta al usuario A y B, pero al banquero no le convino esta transacción. El error pudo ser al revés, quedando el saldo final en 800.

- Postergación o aplazamiento indefinido(a): esto se mencionó en el apartado anterior y consiste en el hecho de que uno o varios procesos nunca reciban el suficiente tiempo de ejecución para terminar su tarea. Por ejemplo, que un proceso ocupe un recurso y lo marque como «ocupado» y que termine sin marcarlo como «desocupado». Si algún otro proceso pide ese recurso, lo verá «ocupado» y esperará indefinidamente a que se «desocupe».
- Condición de espera circular: esto ocurre cuando dos o más procesos forman una cadena de espera que los involucra a todos. Por ejemplo, suponga que el proceso A tiene asignado el recurso «cinta» y el proceso B tiene asignado el recurso «disco». En ese momento al proceso A se le ocurre pedir el recurso «disco» y al proceso B el recurso «cinta». Ahí se forma una espera circular entre esos dos procesos que se puede evitar quitándole a la fuerza un recurso a cualquiera de los dos procesos.
- Condición de no apropiación: esta condición no resulta precisamente de la concurrencia, pero juega un papel importante en este ambiente. Esta condición especifica que si un proceso tiene asignado un recurso, dicho recurso no puede arrebatársele por ningún motivo, y estará disponible hasta que el proceso lo «suelte» por su voluntad.
- Condición de espera ocupada: esta condición consiste en que un proceso pide un recurso que ya está asignado a otro proceso y la condición de no apropiación se debe cumplir. Entonces el proceso estará gastando el resto de su time-slice chequeando si el recurso fue liberado. Es decir, desperdicia su tiempo de ejecución en esperar. La solución más común a este problema consiste en que el sistema operativo se dé cuenta de esta situación y mande a una cola de espera al proceso, otorgándole inmediatamente el turno de ejecución a otro proceso.
- Condición de exclusión mutua: cuando un proceso usa un recurso del sistema realiza una serie de operaciones sobre el recurso y después lo deja de usar. A la sección de código que usa ese recurso se le llama «región crítica».

La condición de exclusión mutua establece que solamente se permite a un proceso estar dentro de la misma región crítica. Esto es, que en cualquier momento solamente un proceso puede usar un recurso a la vez. Para lograr la exclusión mutua se ideó también el concepto de «región crítica».



ca». Para lograr la exclusión mutua generalmente se usan algunas técnicas para lograr entrar a la región crítica: semáforos, monitores, el algoritmo de Dekker y Peterson, los «candados».

- Condición de ocupar y esperar un recurso: consiste en que un proceso pide un recurso y se le asigna. Antes de soltarlo, pide otro recurso que otro proceso ya tiene asignado.

Los problemas descritos son todos importantes para el sistema operativo, ya que debe ser capaz de prevenir o corregirlos. Tal vez el problema más serio que se puede presentar en un ambiente de concurrencia es el «abrazo mortal», también llamado en inglés deadlock. El deadlock consiste en que se presentan al mismo tiempo cuatro condiciones necesarias: la condición de no apropiación, la condición de espera circular, la condición de exclusión mutua y la condición de ocupar y esperar un recurso. Ante esto, si el deadlock involucra a todos los procesos del sistema, el sistema ya no podrá hacer nada productivo. Si el deadlock involucra algunos procesos, éstos quedarán congelados para siempre.

En el área de la informática, el problema del deadlock ha provocado y producido una serie de estudios y técnicas muy útiles, ya que éste puede surgir en una sola máquina o como consecuencia de compartir recursos en una red.

En el área de las bases de datos y sistemas distribuidos han surgido técnicas como el «two phase locking» y el «two phase commit» que van más allá de este trabajo. Sin embargo, el interés principal sobre este problema se centra en generar técnicas para detectar, prevenir o corregir el deadlock.

Las técnicas para prevenir el deadlock consisten en proveer mecanismos para evitar que se presente una o varias de las cuatro condiciones necesarias del deadlock. Algunas de ellas son:

- Asignar recursos en orden lineal: esto significa que todos los recursos están etiquetados con un valor diferente y los procesos solo pueden hacer peticiones de recursos «hacia adelante». Esto es, que si un proceso tiene el recurso con etiqueta «5» no puede pedir recursos cuya etiqueta sea menor que «5». Con esto se evita la condición de ocupar y esperar un recurso.
- Asignar todo o nada: este mecanismo consiste en que el proceso pida todos los recursos que va a necesitar de una vez y el sistema se los da solamente si puede dárselos todos, si no, no le da nada y lo bloquea.
- Algoritmo del banquero: este algoritmo usa una tabla de recursos para saber cuántos recursos tiene de todo tipo. También requiere que los procesos informen del máximo de recursos que van a usar de cada tipo. Cuando un proceso pide un recurso, el algoritmo verifica si, asignándole ese recurso, todavía quedan otros del mismo tipo para que alguno de los procesos en el sistema todavía se le pueda dar hasta su máximo. Si la respuesta es afirmativa, el sistema se dice que está en «estado seguro» y se otorga el recurso. Si la respuesta es negativa, se dice que el sistema está en estado inseguro y se hace esperar a ese proceso.

Para detectar un deadlock, se puede usar el mismo algoritmo del banquero, que aunque no dice que hay un deadlock, sí dice cuándo se está en estado inseguro que es la antesala del deadlock. Sin embargo, para detectar el deadlock se pueden usar las «gráficas de recursos». En ellas se pueden usar cuadrados para indicar procesos y círculos para los recursos, y flechas para indicar si un recurso ya está asignado a un proceso o si un proceso está esperando un recurso.

El deadlock es detectado cuando se puede hacer un viaje de ida y vuelta desde un proceso o recurso. Por ejemplo, suponga los siguientes eventos:

Evento 1: Proceso A pide recurso 1 y se le asigna.

Evento 2: Proceso A termina su time-slice.

Evento 3: Proceso B pide recurso 2 y se le asigna.

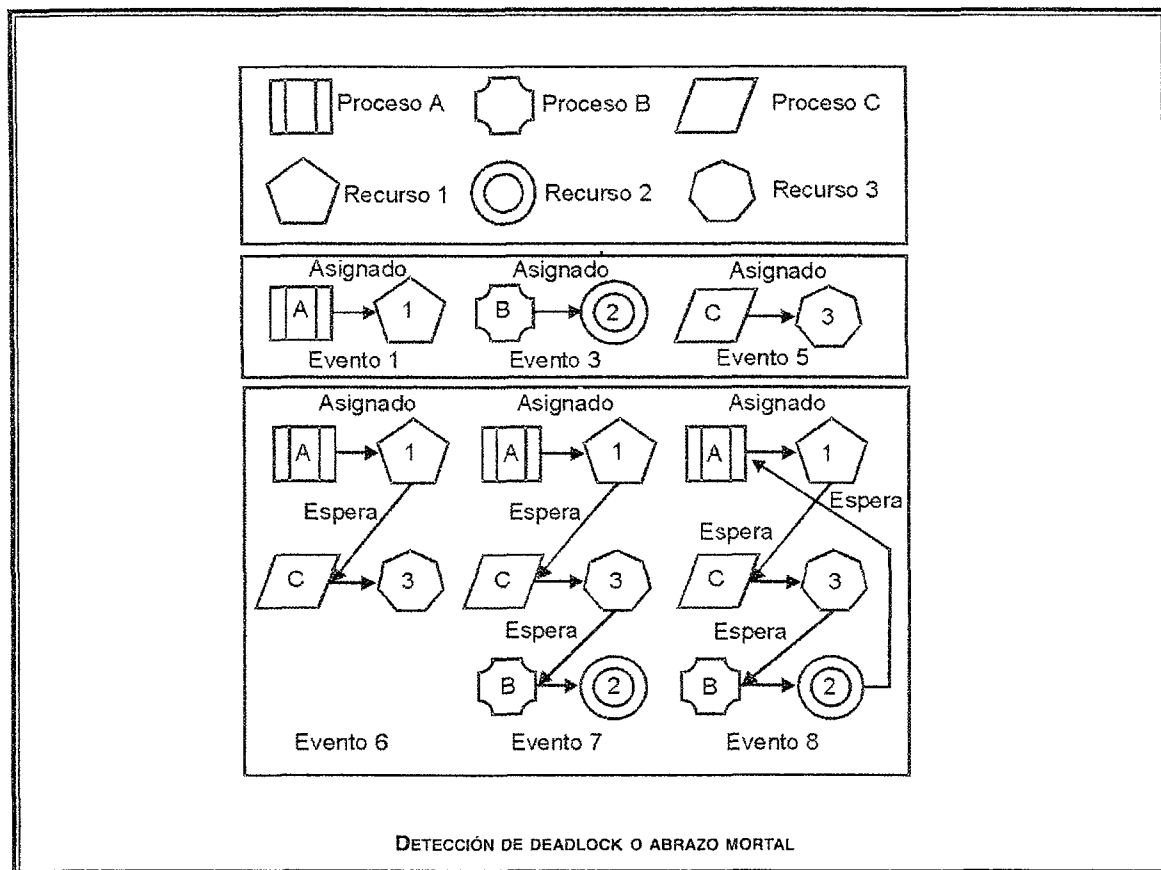
Evento 4: Proceso B termina su time-slice.

Evento 5: Proceso C pide recurso 3 y se le asigna.

Evento 6: Proceso C pide recurso 1 y como lo está ocupando el proceso A espera.

Evento 7: Proceso B pide recurso 3 y se bloquea porque lo ocupa el proceso C.

Evento 8: Proceso A pide recurso 2 y se bloquea porque lo ocupa el proceso B.



En la figura se observa como el gráfico de recursos (Resource Graph) fue evolucionando hasta que se presentó el deadlock, el cual significa que se puede viajar por las flechas desde un proceso o recurso hasta regresar al punto de partida. En el deadlock están involucrados los procesos A, B y C.

Una vez que un deadlock se detecta, es obvio que el sistema está en problemas y lo único que resta por hacer es una de dos cosas: tener algún mecanismo de suspensión o reanudación que permita

copiar todo el contexto de un proceso incluyendo valores de memoria y aspecto de los periféricos que esté usando para reanudarlo en otro momento, o simplemente eliminar un proceso o arrebatárle el recurso, causando para ese proceso la pérdida de datos y tiempo.

## **7. SISTEMAS DE ARCHIVOS.**

Un sistema de archivos (File System) es una estructura de directorios con algún tipo de organización el cual nos permite almacenar, crear y borrar archivos en diferentes formatos. En esta sección se revisarán conceptos importantes relacionados con los sistemas de archivos.

### **7.1. ALMACENAMIENTO FÍSICO DE DATOS.**

En un sistema informático es evidente que existe la necesidad por parte de los usuarios y aplicaciones de almacenar datos en algún medio, a veces por periodos largos y a veces por instantes. Cada aplicación y cada usuario deben tener ciertos derechos con sus datos, como son el poder crearlos y borrarlos, o cambiarlos de lugar; así como tener privacidad frente a otros usuarios o aplicaciones. El subsistema de archivos del sistema operativo se debe encargar de estos detalles, además de establecer el formato físico en el cual almacenará los datos en discos duros, cintas o discos flexibles.

Es de todos conocido que, tradicionalmente, la información en los sistemas modernos se almacena en discos duros, discos flexibles y unidades de disco óptico, y en todos ellos se comparten algunos esquemas básicos para darles formato físico: las superficies de almacenamiento están divididas en círculos concéntricos llamados «pistas» y cada pista dividida en «sectores». A la unión lógica de varias pistas a través de varias superficies «paralelas» de almacenamiento se les llama «cilindros», los cuales son inspeccionados al momento de lectura o escritura de datos por las respectivas unidades físicas llamadas «cabezas». Las superficies de almacenamiento reciben el nombre de «platos» y generalmente están en movimiento rotativo para que las cabezas accedan a las pistas que los componen. Los datos se escriben a través de los sectores en las pistas y cilindros modificando las superficies por medio de las cabezas.

El tiempo que una cabeza necesita para ir de una pista a otra se le llama tiempo de búsqueda y dependerá de la distancia entre la posición actual y la distancia a la pista buscada. El tiempo que tarda una cabeza en posicionarse, dentro de una pista, desde el sector actual al sector deseado se le denomina tiempo de latencia y depende de la distancia entre sectores y la velocidad de rotación del disco. El impacto que tiene las lecturas y escrituras sobre el sistema está determinado por la tecnología usada en los platos y cabezas y por la forma de resolver las peticiones de lectura y escritura, es decir, los algoritmos de planificación.

#### **7.1.1. Algoritmos de planificación de peticiones.**

Los algoritmos de planificación de peticiones de lectura y escritura a discos se encargan de registrar dichas peticiones y de responderlas en un tiempo razonable. Los algoritmos más comunes para esta tarea son:

- **Primero en llegar, primero en ser servido (FIFO):** las peticiones son encoladas de acuerdo al orden en que llegaron y de esa misma forma se van leyendo o escribiendo las mismas. La ventaja de este algoritmo es su simplicidad que no causa sobrecarga, su desventaja principal es que no aprovecha ninguna característica de las peticiones, de manera que es muy factible que el brazo del disco se mueva muy ineficientemente, ya que las peticiones pueden tener direcciones en el disco unas muy alejadas de otras. Por ejemplo, si se están haciendo peticiones a los sectores 6, 10, 8, 21 y 4, las mismas serán resueltas en el mismo orden.

- Primero el más cercano a la posición actual: en este algoritmo las peticiones se ordenan de acuerdo a la posición actual de la cabeza lectora, sirviendo primero a aquellas peticiones más cercanas y reduciendo, así, el movimiento del brazo, lo cual constituye la ventaja principal de este algoritmo. Su desventaja consiste en que puede haber solicitudes que se queden esperando para siempre, en el infortunado caso de que existan peticiones muy alejadas y en todo momento estén entrando peticiones que estén más cercanas. Para las peticiones 6, 10, 8, 21 y 4, las mismas serán resueltas en el orden 4, 6, 8, 10 y 21.
- Por exploración (algoritmo del elevador): en este algoritmo el brazo se está moviendo en todo momento desde el perímetro del disco hacia su centro y viceversa, resolviendo las peticiones que existan en la dirección que tenga en turno. En este caso las peticiones 6, 10, 8, 21 y 4 serán resueltas en el orden 6, 10, 21, 8 y 4; es decir, la posición actual es 6 y como va hacia los sectores de mayor numeración (hacia el centro, por ejemplo), en el camino sigue el sector 10, luego el 21, siendo el más céntrico. Ahora el brazo resolverá las peticiones en su camino hacia el borde del plato y la primera petición que se encuentra es la del sector 8 y luego la 4. La ventaja de este algoritmo es que el brazo se mueve mucho menos que en FIFO y evita la espera indefinida; su desventaja es que no es justo, ya que no sirve las peticiones en el orden en que llegan, además las peticiones en los extremos interior y exterior gozan de un tiempo de respuesta un poco mayor.
- Por exploración circular: es una variación del algoritmo anterior, con la única diferencia que al llegar a la parte central, el brazo regresa al exterior sin resolver ninguna petición, lo cual proporciona un tiempo de respuesta más cercano al promedio para todas las peticiones, sin importar si están cercas del centro o del exterior.

### 7.1.2. Asignación del espacio de almacenamiento.

El subsistema de archivos se debe encargar de localizar espacio libre en los medios de almacenamiento para guardar archivos y para después borrarlos, renombrarlos o agrandarlos. Para ello se vale de ubicaciones especiales que contienen la lista de archivos creados y por cada archivo una serie de direcciones que apuntan a su contenido. Esas localidades especiales se llaman directorios. Para asignarle espacio a los archivos existen tres criterios generales que se describen a continuación.

- Asignación contigua: cada directorio contiene los nombres de archivos y la dirección del bloque inicial de cada archivo, así como el tamaño total de cada uno. Por ejemplo, si un archivo comienza en el sector 17 y mide 10 bloques, cuando el archivo sea accedido, el brazo se moverá inicialmente al bloque 17 y de ahí hasta el 27. Si el archivo es borrado y luego creado otro más pequeño, quedarán huecos inútiles entre archivos útiles, lo cual se denomina fragmentación externa.
- Asignación encadenada: con este criterio los directorios contienen los nombres de archivos y por cada uno de ellos la dirección del bloque inicial que compone al archivo. Cuando un archivo es leído, el brazo va a esa dirección inicial y encuentra los datos iniciales junto con la dirección del siguiente bloque y así sucesivamente.

Con este criterio no es necesario que los bloques estén contiguos y no existe la fragmentación externa, pero en cada «eslabón» de la cadena se desperdicia espacio con las direcciones mismas. En otras palabras, lo que se crea en el disco es una lista ligada.

- Asignación con índices (indexada): en este esquema se guarda en el directorio un bloque de índices para cada archivo, con apuntadores hacia todos sus bloques constituyentes, de manera

que el acceso directo se agiliza notablemente, a cambio de sacrificar varios bloques para almacenar dichos apuntadores. Cuando se quiere leer un archivo o cualquiera de sus partes, se hacen dos accesos: uno al bloque de índices y otro a la dirección deseada. Éste es un esquema excelente para archivos grandes pero no para pequeños, para los que, la relación entre bloques destinados a índices respecto de los destinados para datos supone un coste inasumible.

### 7.1.3. Métodos de acceso en los sistemas de archivos.

Los métodos de acceso se refieren a las capacidades que el subsistema de archivos proporciona para acceder a datos dentro de los directorios y medios de almacenamiento en general. Se ubican tres formas generales: acceso secuencial, acceso directo y acceso directo indexado.

- **Acceso secuencial:** es el método más lento y consiste en recorrer los componentes de un archivo uno a uno hasta llegar al registro deseado. Se necesita que el orden lógico de los registros sea igual al orden físico en el medio de almacenamiento. Este tipo de acceso se usa comúnmente en cintas y cartuchos.
- **Acceso directo:** permite acceder a cualquier sector o registro inmediatamente, por medio de llamadas al sistema como la de seek. Este tipo de acceso es rápido y se usa comúnmente en discos duros y discos o archivos manejados en memoria de acceso aleatorio.
- **Acceso directo indexado:** este tipo de acceso es útil para grandes volúmenes de información o datos. Consiste en que cada archivo tiene una tabla de apuntadores, donde cada apuntador va a la dirección de un bloque de índices, lo cual permite que el archivo se expanda a través de un espacio enorme. Consume una cantidad importante de recursos en las tablas de índices pero es muy rápido.

### 7.1.4. Operaciones soportadas por el subsistema de archivos.

Independientemente de los algoritmos de asignación de espacio, de los métodos de acceso y de la forma de resolver las peticiones de lectura y escritura, el subsistema de archivos debe proveer un conjunto de llamadas al sistema para operar con los datos proporcionando mecanismos de protección y seguridad. Las operaciones básicas que la mayoría de los sistemas de archivos soportan son:

- **Crear (create):** permite crear un archivo sin datos, con el propósito de indicar que ese nombre ya está usado. Se deben crear las estructuras básicas para soportarlo.
- **Borrar (delete):** eliminar el archivo y liberar los bloques para su uso posterior.
- **Abrir (open):** antes de usar un archivo se debe abrir para que el sistema conozca sus atributos, tales como el propietario, la fecha de modificación, etc.
- **Cerrar (close):** después de realizar todas las operaciones deseadas, el archivo debe cerrarse para asegurar su integridad y para liberar recursos de su control en la memoria.
- **Leer o Escribir (read, write):** añadir información al archivo o leer el carácter o una cadena de caracteres a partir de la posición actual.
- **Concatenar (append):** es una forma restringida de la llamada «write», en la cual sólo se permite añadir información al final del archivo.

- Localizar (seek): para los archivos de acceso directo se permite posicionar el apuntador de lectura o escritura en un registro aleatorio, a veces a partir del inicio o final del archivo.
- Leer atributos: permite obtener una estructura con todos los atributos del archivo especificado, tales como permisos de escritura, de borrado, ejecución, etc.
- Poner atributos: permite cambiar los atributos de un archivo, por ejemplo en UNIX, donde todos los dispositivos se manejan como si fueran archivos, es posible cambiar el comportamiento de una terminal con una de estas llamadas.
- Renombrar (rename): permite cambiarle el nombre e incluso a veces la posición en la organización de directorios del archivo especificado.

Los subsistemas de archivos también proporcionan un conjunto de llamadas para operar sobre directorios, las más comunes son crear, borrar, abrir, cerrar, renombrar y leer. Sus funcionalidades son obvias, pero existen también otras dos operaciones no tan comunes que son la de «crear un enlace» y la de «destruir un enlace». La operación de crear un enlace sirve para que, desde diferentes puntos de la organización de directorios, se pueda acceder a un mismo archivo sin necesidad de copiarlo o duplicarlo. La llamada a «destruir un enlace» lo que hace es eliminar esas referencias, sin afectar al archivo original.

#### 7.1.5. Algunas facilidades extras de los sistemas de archivos.

Algunos sistemas de archivos proporcionan herramientas al administrador del sistema para facilitarle su labor. Las más notables son la utilidad de compartir archivos y los sistemas de «cuotas».

La utilidad de compartir archivos se refiere a la posibilidad de que los permisos de los archivos o directorios dejen que un grupo de usuarios puedan acceder para diferentes operaciones: leer, escribir, borrar, crear, etc. El dueño verdadero es quien decide qué permisos se aplicarán al grupo e, incluso, a otros usuarios que no formen parte de su grupo. La utilidad de «cuotas» se refiere a que el sistema de archivos es capaz de llevar un control para que cada usuario pueda usar un máximo de espacio de almacenamiento. Cuando el usuario excede ese límite, el sistema le avisa y le niega el permiso de seguir escribiendo, obligándole a borrar algunos archivos si es que quiere almacenar otros o que crezcan los existentes. La versión de UNIX SunOS contiene esa utilidad.

#### 7.2. SISTEMAS DE ARCHIVOS AISLADOS.

Los sistemas de archivos aislados son aquellos que residen en un solo ordenador y no existe la posibilidad de que, aún estando en una red, otros sistemas puedan usar sus directorios y archivos. Por ejemplo, los archivos en discos duros en el sistema MS-DOS clásico se puede englobar en esta categoría.

#### 7.3. SISTEMAS DE ARCHIVOS COMPARTIDOS O DE RED.

Estos sistemas de archivos permiten ser accedidos desde otros nodos en una red. Generalmente, existe un «servidor» que es el ordenador en donde reside el sistema de archivos físicamente, y por otro lado están los «clientes», que se valen del servidor para ver sus archivos y directorios como si fueran locales a la máquina.

Algunos autores llaman a estos sistemas de archivos «sistemas de archivos distribuidos» lo cual no se va a discutir en este trabajo.

Los sistemas de archivos compartidos en red más populares son los provistos por Netware, el Remote File Sharing (RFS en UNIX), Network File System (NFS de Sun Microsystems) y el Andrew File System (AFS). En general, los servidores proporcionan un medio para que los clientes, localmente, realicen peticiones de operaciones sobre archivos las cuales son «atrapadas» por un «driver» o un «módulo» en el núcleo del sistema operativo, el cual se comunica con el servidor a través de la red, ejecutándose la operación en el servidor. Existen servidores de tipo «stateless» y «no-stateless». Un servidor «stateless» no registra el estado de las operaciones sobre los archivos, de manera que el cliente se encarga de todo ese trabajo. La ventaja de este esquema es que, si el servidor falla, el cliente no perderá información ya que ésta se guarda en memoria localmente, de manera que cuando el servidor reanude su servicio el cliente proseguirá como si nada hubiese sucedido. Con un servidor «no-stateless», esto no es posible.

La protección sobre las operaciones se lleva a cabo tanto en los clientes como en el servidor: si el usuario quiere ejecutar una operación indebida sobre un archivo, recibirá un mensaje de error y posiblemente se envíe un registro al subsistema de «seguridad» para informar al administrador del sistema de dicho intento de violación.

En la práctica, el conjunto de permisos que cada usuario tiene sobre el total de archivos se almacena en estructuras llamadas «listas de acceso» (access lists).

#### 7.4. TENDENCIAS ACTUALES.

Con el gran auge de las redes de comunicaciones y su incremento de ancho de banda, la proliferación de paquetes que ofrecen la compartición de archivos es común. Los esquemas más solicitados en la industria son el poder acceder a grandes volúmenes de información, que residen en grandes servidores, desde los ordenadores personales y desde otros servidores también.

A veces se requieren soluciones más complejas con ambientes heterogéneos: diferentes sistemas operativos y diferentes arquitecturas. Uno de los sistemas de archivos más expandidos en estaciones de trabajo es el NFS, y prácticamente todas las versiones de UNIX traen instalado un cliente y hasta un servidor de este servicio. Es posible así que una gran cantidad de ordenadores personales accedan a grandes volúmenes de información desde una sola estación de trabajo, e incluso tener la flexibilidad de usar al mismo tiempo servidores de Novell y NFS. Soluciones similares se dan con algunos otros paquetes comerciales. Lo importante aquí es observar que el mundo se va moviendo hacia soluciones distribuidas, y hacia la estandarización que, muchas veces, es «de facto».

### 8. PRINCIPIOS EN EL MANEJO DE LA ENTRADA/SALIDA.

El código destinado a manejar la entrada y salida de los diferentes periféricos en un sistema operativo es de una extensión considerable y sumamente complejo. Resuelve las necesidades de sincronizar, atrapar interrupciones y ofrecer llamadas al sistema para los programadores. En esta sección se repasarán los principios más importantes a tomar en cuenta en este módulo del sistema operativo.

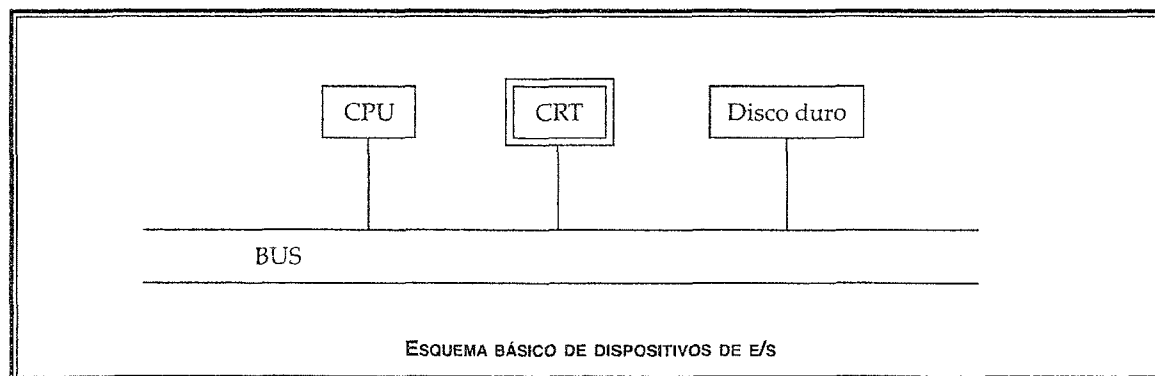
## 8.1. DISPOSITIVOS DE ENTRADA/SALIDA.

Los dispositivos de entrada salida se dividen, en general, en dos tipos: dispositivos orientados a bloque y dispositivos orientados a carácter. Los dispositivos orientados a bloque tienen la propiedad de que se pueden direccionar, esto es, el programador puede escribir o leer cualquier bloque del dispositivo realizando primero una operación de posicionamiento sobre él. Los dispositivos más comunes orientados a bloque son los discos duros, la memoria, discos compactos y, posiblemente, unidades de cinta. Por otro lado, los dispositivos orientados a carácter son aquellos que trabajan con secuencias de bytes sin importar su longitud ni ninguna agrupación en especial. No son dispositivos direccionables. Ejemplos de estos dispositivos son el teclado, la pantalla o display y las impresoras.

La clasificación anterior no es perfecta, porque existen varios dispositivos que generan entrada o salida que no pueden englobarse en esas categorías. Por ejemplo, un reloj que genera pulsos. Sin embargo, aunque existan algunos periféricos que no se puedan categorizar, todos están administrados por el sistema operativo por medio de una parte electromecánica y una parte de software.

## 8.2. CONTROLADORES DE DISPOSITIVOS (TERMINALES Y DISCOS DUROS).

Los controladores de dispositivos (también llamados adaptadores de dispositivos) son la parte electrónica de los periféricos, los cuales pueden tener la forma de una tarjeta o un circuito impreso integrado en la placa base del ordenador o venderse por separado insertándose en una ranura diseñada al efecto.



Los controladores de dispositivos generalmente trabajan con voltajes de 5 y 12 volts con el dispositivo propiamente, y con el ordenador a través de interrupciones. Estas interrupciones viajan por el «bus» del ordenador y son recibidas por la CPU la cual a su vez pondrá en ejecución algún programa que sabrá qué hacer con esa señal. A ese programa se le llama «manejador de dispositivo» (device driver). Algunas veces el mismo controlador contiene un pequeño programa en una memoria de solo lectura o en memoria de acceso aleatorio, no volátil y reescribible, que interactúa con el correspondiente manejador en el ordenador. En la figura se muestra un esquema simple de dispositivos orientados a bloque y otros a carácter.

Para intercambiar datos o señales entre el ordenador y los controladores, muchas veces se usan registros o secciones predefinidas de la memoria del ordenador. A este esquema se le llama «gestión de entrada/salida mapeado por memoria» (memory mapped I/O). Por ejemplo, para un PC IBM se muestran los vectores de interrupción y las direcciones para la Entrada/Salida en la tabla siguiente.



CONTROLADOR	DIRECCIÓN (HEX)	VECTOR DE INTERRUPCIÓN
Reloj	040 - 043	8
Teclado	060 - 063	9
Disco duro	320 - 32F	13
Impresora	378 - 37F	15
Monitor mono	380 - 3BF	–
Monitor color	3D0 - 3DF	–
Disco flexible	3F0 - 3F7	14

### 8.3. ACCESO DIRECTO A MEMORIA (DMA).

El acceso directo a memoria se inventó con el propósito de liberar a la CPU de la carga de atender a algunos controladores de dispositivos. Para comprender su funcionamiento vale la pena revisar cómo trabaja un controlador sin DMA: cuando un proceso requiere algunos bloques de un dispositivo, se envía una señal al controlador con la dirección del bloque deseado. El controlador lo recibe a través del «bus» y el proceso puede estar esperando la respuesta (trabajo síncrono) o puede estar haciendo otra cosa (trabajo asíncrono). El controlador recibe la señal y lee la dirección del bus. Envía a su vez una o varias señales al dispositivo mecánico (si es que lo hay) y espera los datos. Cuando los recibe los escribe en un buffer local y envía una señal a la CPU indicándole que los datos están listos. La CPU recibe esta interrupción y comienza a leer byte por byte o palabra por palabra los datos del buffer del controlador (a través del device driver) hasta terminar la operación.

Como se ve, la CPU gasta varios ciclos en leer los datos deseados. El DMA soluciona ese problema de la manera siguiente: cuando un proceso requiere uno o varios bloques de datos, la CPU envía al controlador de DMA la petición junto con el número de bytes deseados y la dirección en dónde quiere que se almacenen de regreso. El DMA actuará como una «cpu secundaria» en cuanto a que tiene el poder de tomar el control del «bus» e indicarle al verdadero CPU que espere. Cuando el controlador tiene listos los datos, el DMA «escucha» si el «bus» está libre aprovechando esos ciclos para ir leyendo los datos del buffer del controlador e ir escribiéndolos en el área de memoria que la CPU le indicó.

Una vez que todos los datos han sido escritos en el destino, se le envía una interrupción a la CPU para que use los datos. El ahorro con el DMA es que la CPU ya no es interrumpida (aunque sí puede ser retardada por el DMA) salvando así el «cambio de contexto» y además el DMA aprovechará aquellos ciclos en que el «bus» no es usado por la CPU.

El hecho de que los controladores necesiten buffers internos se debe a que, conforme ellos reciben datos de los dispositivos que controlan, los deben poder almacenar temporalmente, pues la CPU no está lista en todo momento para leerlos.

### 8.4. PRINCIPIOS EN EL SOFTWARE DE ENTRADA/SALIDA.

Los principios de software en la E/S se resumen en cuatro puntos: el software debe ofrecer manejadores de interrupciones, manejadores de dispositivos, software que sea independiente de los dispositivos y software para usuarios.

#### **8.4.1. Manejadores de interrupciones.**

El primer objetivo referente a los manejadores de interrupciones consiste en que el programador o el usuario no debe darse cuenta de los manejos de bajo nivel para los casos en que el dispositivo esté ocupado y se deba suspender el proceso o sincronizar algunas tareas. Desde el punto de vista del proceso o usuario, el sistema simplemente tardó más o menos en responder a su petición.

#### **8.4.2. Manejadores de dispositivos.**

El sistema debe proporcionar los manejadores de dispositivos necesarios para los periféricos, así como ocultar las peculiaridades del manejo interno de cada uno de ellos, tales como el formato de la información, los medios mecánicos, los niveles de voltaje y otros. Por ejemplo, si el sistema tiene varios tipos diferentes de discos duros, para el usuario o programador las diferencias técnicas existentes entre ellos no le deben importar. Los manejadores le deben ofrecer el mismo conjunto de rutinas para leer y escribir datos.

#### **8.4.3. Software independiente del dispositivo.**

Este es un nivel superior de independencia que el ofrecido por los manejadores de dispositivos. Aquí el sistema operativo debe ser capaz, en la medida de lo posible, de ofrecer un conjunto de utilidades para acceder a periféricos o programarlos de una manera consistente. Por ejemplo, que para todos los dispositivos orientados a bloque se disponga de una llamada para decidir si se desea usar «buffers» o no, o para posicionarse en ellos.

#### **8.4.4. Software para usuarios.**

La mayoría de las rutinas de Entrada/Salida trabajan en modo privilegiado o son llamadas al sistema, que se ligan a los programas del usuario formando parte de sus aplicaciones y que no le dejan ninguna flexibilidad en cuanto a la apariencia de los datos. Existen otras librerías en donde el usuario sí tiene poder de decisión (por ejemplo la llamada a «printf» en el lenguaje «C»). Otra facilidad ofrecida son las áreas de encolado de trabajos (spooling areas), tales como las de impresión y correo electrónico.

### **8.5. RELOJES.**

Los relojes son esenciales para el buen funcionamiento de cualquier sistema porque juegan un papel decisivo en la sincronización de procesos, en la programación de trabajos por lotes y para la asignación de turnos de ejecución entre otras tareas relevantes. Generalmente se cuenta con dos relojes en el sistema: uno que lleva la hora y fecha del sistema y que oscila entre 50 y 60 veces por segundo y el reloj que oscila entre 5 y 3.000 millones de veces por segundo y que se encarga de enviar interrupciones a la CPU de manera periódica. El reloj de mayor frecuencia sirve para controlar el tiempo de ejecución de los procesos, para despertar los procesos que están «durmiendo» y para lanzar o iniciar procesos que fueron programados.

Para mantener la hora y fecha del sistema generalmente se usa un registro alimentado por una pila de alta duración que almacena estos datos y que se programan de fábrica por primera vez. Así, aunque se suspenda la energía la fecha permanece. Para lanzar procesos (chequeo de tiempo ocioso de un dispositivo, terminación del time-slice de un proceso, etc.), se almacena un valor en un registro (QUANTUM) el cual se decrementa con cada ciclo del reloj, y cuando llega a cero se dispara un proceso que ejecutará las operaciones necesarias (escoger un nuevo proceso en ejecución, verificar el funcionamiento del motor del disco flexible, hacer eco de un carácter del teclado, etc.).

## 8.6. PLUG AND PLAY.

Si bien durante años la arquitectura interna de los miniordenadores e incluso las del PC de IBM original, se apoyaban en un solo Bus de comunicaciones, a medida que aumentó la rapidez de los procesadores y las memorias, la capacidad de este bus se sometió a una demanda excesiva. Para corregirlo se añadieron más buses, tanto para los dispositivos de E/S más rápidos, como para el tráfico entre la CPU y la memoria. Como consecuencia de esta evolución, en la actualidad un sistema Pentium grande tiene al menos nueve buses (caché; memoria; PCI; AGP; USB; IDE; SCSI e ISA), todos con diferente tasa de transferencia y función.

Para operar en un entorno tan variado y complejo, el sistema operativo tiene que saber qué dispositivos hay y cómo configurarlos. Este requisito llevó a Intel y Microsoft a diseñar un sistema para PC denominado Plug and Play (Conectar y Usar), basado en un concepto similar al que se implementó por primera vez en la Apple Macintosh.

Antes de Plug and Play, cada tarjeta de E/S tenía un nivel fijo de solicitud de interrupción y direcciones fijas para sus registros de E/S. Los problemas se presentaban cuando un usuario adquiría alguna ampliación (p. ej. una tarjeta de sonido y un modem) y resultaba que ambas coincidían en alguno de sus parámetros de configuración (p. ej. la misma interrupción). La solución fue incluir interruptores o puentes (jumpers) en cada tarjeta de E/S, haciendo que ésta admitiera varias configuraciones diferentes y explicando al usuario cómo ajustarlos para evitar conflictos con otras tarjetas instaladas. Esta situación convertía la instalación y puesta en marcha de nuevas capacidades en un calvario reservado a unos pocos expertos.

Plug and Play (cuyo acrónimo es PnP) vino a poner orden en el caos de la configuración de dispositivos de E/S. Con PnP, el sistema recaba información de manera automática sobre los dispositivos de E/S, asigna de modo centralizado niveles de interrupción y direcciones de E/S y luego comunica a cada tarjeta su asignación, configurándose esta de acuerdo a lo establecido. En cada arranque del sistema la configuración es susceptible de modificarse si se precisa. A grandes rasgos el sistema funciona como sigue en un Pentium:

Todo Pentium tiene una placa base (o Motherboard) con un programa denominado Sistema Básico de Entrada/Salida BIOS (Basic Input Output System). El BIOS contiene software de E/S de bajo nivel que incluye procedimientos para leer el teclado, escribir en la pantalla y efectuar E/S de disco entre otras cosas.

Cuando el ordenador arranca se inicia el BIOS que, en primer lugar, determina la memoria RAM instalada y si el teclado y otros dispositivos básicos están operativos. A continuación se exploran los buses ISA y PCI para detectar todos los dispositivos conectados a ellos. Entre los dispositivos detectados estarán los del tipo Plug and Play y los heredados (diseñados antes de inventarse el PnP) que tienen interrupciones y direcciones de E/S fijas. Todos los dispositivos se registran y si no son los mismos que había en el último arranque del sistema, se configuran los nuevos.

En un paso posterior, la BIOS localiza un dispositivo de arranque entre los disponibles en una lista (usualmente el disco flexible, una unidad de CD-ROM, una unidad de Disco Duro y, en ciertos sistemas más modernos, un dispositivo USB o un dispositivo externo a través de una Red de Área Local). Localizado el dispositivo de arranque, se lee el primer sector, se almacena en memoria y se ejecuta. Este sector contiene un programa que, por lo general, examina la tabla de particiones al final del sector de arranque para determinar la partición activa, luego se lee un cargador de arranque secundario de esa partición; el cargador lee el sistema operativo de la partición activa y lo pone en marcha.

Una vez puesto en marcha, el sistema operativo consulta el BIOS para obtener información de configuración; verifica que cada dispositivo cuente con el controlador de dispositivo correspondiente (si fal-

ta alguno pide al usuario insertar el disquete, CD-ROM o señalar la ruta, donde encontrar el controlador de dispositivo proporcionado por el fabricante del mismo). Cuando el sistema operativo tiene todos los controladores de dispositivo los carga en el Kernel; asigna los valores iniciales a sus tablas; crea los procesos en segundo plano necesarios y arranca un programa de inicio de sesión o de GUI en cada terminal.

En la vida real, Plug and Play no siempre funciona correctamente, hasta el punto que algunos lo denominan Plug and Pray -Conectar y Rezar-.

La especificación Plug and Play no es una propiedad o característica exclusiva del sistema operativo. Para que un sistema sea PnP deben estar implicados un conjunto de diferentes colectivos: el suministrador del sistema operativo; el fabricante del hardware del sistema; el desarrollador de la BIOS y los fabricantes de dispositivos. Para conseguir la coordinación necesaria debe existir un conjunto bien definido de interfaces y delegaciones claras de responsabilidad para alcanzar las metas previstas.

Debemos recordar que todo el subsistema PnP concierne principalmente a la gestión de cuatro tipos de recursos diferentes en nombre de los dispositivos:

- Memoria: los requisitos de memoria física del dispositivo (p.ej. cuántas páginas de memoria necesita el dispositivo y cualquier restricción de alineamiento).
- E/S: los puertos de E/S a los que responderá el dispositivo. La información de configuración del dispositivo incluye una especificación de cada uno de los conjuntos alternativos de puertos que el dispositivo puede utilizar (si los hay).
- DMA: cualquier canal DMA que requiera el dispositivo y cualquier canal alternativo que pueda utilizar.
- IRQ: los requisitos de IRQ del dispositivo, IRQ alternativos y si el dispositivo puede o no compartir una IRQ con otros dispositivos.

Merece la pena aclarar que la especificación Plug and Play nació para permitir la configuración de dispositivos conectados en el bus ISA de un ordenador. El bus PCI ya nació comportándose como Plug and Play, aunque la especificación PCI no emplea en ningún momento el término Plug and Play, su hardware implementa lo que, hoy en día, se conoce como Plug and Play.

## 8.7. LA INTERFAZ DE USUARIO.

Todo ordenador debe disponer de un sistema capaz de interactuar con el usuario del mismo, mediante el cual puedan introducirse órdenes y recibir la información generada de una forma adecuada e inteligible, estableciéndose una comunicación hombre-máquina.

Al sistema compuesto por elementos hardware y software adecuados a este fin, se le conoce como la interfaz de usuario.

La interfaz de usuario debe proporcionar la capacidad de comunicación bidireccional. Hoy en día, cualquier ordenador de uso general dispone por lo menos de un teclado, como canal de comunicación hombre-máquina y de una pantalla o monitor como canal máquina-hombre, constituyendo el conjunto pantalla-teclado la interfaz hardware más extendida y usada en todos los sistemas.

Además del teclado y la pantalla, existe una gran variedad de sistemas que amplían la capacidad de la interfaz de usuario o la especializan para ciertas necesidades concretas. El ratón, la tableta digita-

lizadora, el reconocimiento de voz a través de un micrófono, la captura de imágenes mediante cámaras, los guantes táctiles o PowerGlove (también conocidos como guantes de datos), que permiten el reconocimiento de signos y la interacción con sistemas multimedia, son ejemplos de sistemas de entrada de datos. Las impresoras, terminales braille, los sintetizadores de voz y otros más novedosos o experimentales como los generadores de aromas (basados en la existencia de ciertas sustancias químicas básicas, a partir de las cuales es posible reproducir cualquier olor, de manera similar a cómo los monitores reproducen cualquier color como recombinación de los tres colores básicos Rojo, Verde y Azul) son ejemplos de sistemas de salida de datos.

Toda interfaz hardware se complementa con una interfaz software que aporta en gran medida la funcionalidad y comunicabilidad definitiva, estableciendo una filosofía o modo de interacción con el usuario que puede llegar a ser radicalmente distinta aun empleando los mismos interfaces físicos.

Históricamente, la comunicación hombre-máquina ha consistido en la introducción de comandos a través del teclado, que son ejecutados por el ordenador, mostrando por la pantalla el resultado de dicha ejecución. El programa encargado de aceptar las órdenes tecleadas y ejecutar las tareas asociadas se denomina intérprete de comandos. UNIX, LINUX y MS-DOS son ejemplos de sistemas operativos con intérprete de comandos.

Esta forma de comunicación está orientada a carácter, esto es, se envían los caracteres tecleados al ordenador y éste remite la información al usuario carácter a carácter, organizándose en la pantalla, como en un papel, en líneas consecutivas.

Con la mejora en la potencia de los ordenadores, ha sido posible el que, hoy en día, casi todos los ordenadores personales empleen la denominada Interfaz Gráfica de Usuario GUI (Graphical User Interface).

La GUI fue inventada por Douglas Engelbart y su grupo de investigación en el Stanford Research Institute. Posteriormente la copiaron los investigadores de Xerox para sus sistemas. Steve Jobs, uno de los fundadores de Apple, visitando el centro de investigación de Xerox, vio una GUI en uno de sus ordenadores, sirviéndole de idea para un nuevo modelo de ordenador: la Apple Lisa. Esta máquina resultó demasiado cara, siendo un fracaso comercial. Sin embargo, su sucesora la Apple Macintosh tuvo un éxito sin precedentes y ha constituido *de facto*, la inspiración (sino el molde) de otros sistemas basados en GUI, como lo es Microsoft Windows.

Una GUI tiene cuatro elementos indispensables, conocidos por su acrónimo WIMP (Windows, Icons, Menus, Pointer device) Ventanas, Iconos, Menús y un dispositivo apuntador.

Las ventanas son zonas rectangulares de área de pantalla, en las que se ejecutan programas. Los iconos son pequeños símbolos sobre los que se puede apuntar y seleccionar para ejecutar alguna acción. Los menús son listas de acciones disponibles, presentadas en formato textual, en un determinado momento y programa entre las que el usuario puede escoger. Por último, el dispositivo apuntador es un elemento hardware que sirve para desplazar un cursor por la pantalla y seleccionar cosas. El exponente más conocido de este tipo de dispositivos es el ratón aunque también cumplen con esta función otros dispositivos no menos conocidos, como las pantallas táctiles, las tabletas digitalizadoras y, más profusamente, los denominados touchpads (superficies deslizantes sensibles a la presión) presentes en la mayoría de los actuales ordenadores portátiles, que han sustituido a la tradicional bola-ratón de los primeros equipos.

El software de salida necesario para implementar una interfaz GUI es de una enorme complejidad, hasta el punto que, en el caso de Windows, constituye la parte predominante del Sistema Operativo. Existen en la actualidad dos corrientes principales sobre la forma de implementar un GUI y su interacción con el sistema operativo: la representada por los sistemas operativos Windows, en donde la GUI está incorporada en el núcleo del sistema operativo y es indisoluble de él, y los siste-

mas UNIX y LINUX que han apostado por un modelo independiente del núcleo del sistema operativo. En la primera opción, se consigue una interfaz muy optimizada y potente, por su completa imbricación en el núcleo del S.O.; En la segunda opción, se consigue una enorme estabilidad de los sistemas pues, la caída del subsistema de interfaz solamente afecta a un usuario, que puede reiniciar el mismo como si se tratara de cualquier otra aplicación, preservando el núcleo que permanece plenamente operativo en todo momento. Otra gran ventaja de los sistemas GUI independientes del núcleo del S.O. es la posibilidad de disponer de más de un GUI, suministrados por terceros, con lo que se fomenta la existencia de un mercado de subsistemas GUI, adaptables a entornos especializados y en gran medida configurables.

## 9. VINCULACIÓN E INCRUSTACIÓN DE OBJETOS.

### 9.1. HISTORIA Y GENERALIDADES.

Microsoft creó la tecnología OLE para resolver de una forma parecida a la orientación a objetos, denominada software de componentes, muchos problemas prácticos encontrados en sus años de desarrollo de sistemas operativos y aplicaciones. OLE proporciona las especificaciones necesarias y los servicios principales para habilitar el software de componentes.

Los interfaces gráficos de usuario popularizaron la metáfora del portapapeles, con las operaciones de «cortar», «copiar» y «pegar» que simplificaron enormemente la creación de documentos compuestos (compound documents), que incluyen texto, gráficos y otros tipos de contenidos.

Aunque el portapapeles trabajaba bien, tenía limitaciones que llevaron a Microsoft a avanzar un paso más y crear un complejo protocolo de intercambio dinámico de datos (DDE Dynamic Data Exchange), para simplificar y ampliar la capacidad de creación y gestión de documentos compuestos. Alrededor del protocolo DDE creció la versión 1.0 de OLE en 1991, que se presentó a los desarrolladores como un estándar. OLE 1.0 amplió enormemente la capacidad de creación y gestión de estos documentos compuestos. Uno coloca objetos incrustados u objetos vinculados en un documento, que almacena los datos iniciales empleados para crearlos o un vínculo a esos datos, así como información sobre el formato.

El acrónimo OLE es una abreviatura de «Object Linking and Embedding» Vinculación e Incrustación de Objetos. Toda la complejidad de editar el contenido se reduce a pulsar con el ratón sobre el área destinada a mostrar ese contenido y los datos del objeto vuelven a estar disponibles, de forma automática, en su editor original.

Los diseñadores de OLE 1.0 comprendieron que los objetos documentos compuestos eran en realidad un caso específico de «software de componentes» (pequeños elementos de software que pueden conectarse en una aplicación, extendiendo de ese modo su funcionalidad sin requerir cambios adicionales). En el paradigma de los documentos compuestos, el editor del documento es un contenedor genérico que puede contener cualquier tipo de objeto contenido. Uno puede entonces incorporar gráficos, sonido, vídeo, imágenes y muchos otros tipos de componentes en el documento contenedor, sin necesidad de actualizar el mismo.

En un sentido más amplio, el software de componentes tiene una mayor aplicación que solamente en los documentos compuestos. Es un modelo de conectores multipropósito más potente y flexible que otros modelos, como las librerías de enlace dinámico (DLL); los controles Visual Basic o similares.

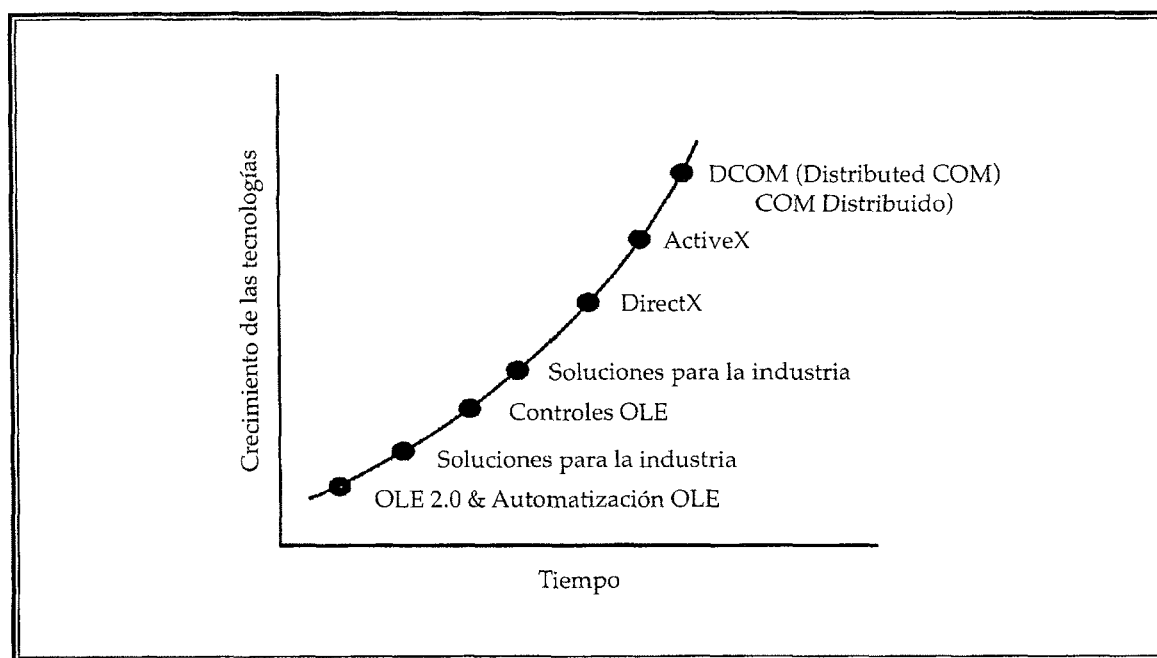
Con OLE 2.0, desarrollado en 1993, se construyó una vasta infraestructura de soporte del software de componentes en un amplio abanico de niveles de complejidad.

El núcleo de esta infraestructura es una arquitectura potente y ampliable denominada COM (Component Object Model) Modelo de Objetos Componente. Es dentro de COM donde se da respuesta a los problemas de software más desconcertantes, incluyendo aquellos de las arquitecturas de servicio ampliable, objetos fuera de los límites de una aplicación y el versionado. Más aún, estas soluciones tienen que ver con componentes binarios en un sistema en ejecución, en lugar de componentes en código fuente dentro de una aplicación.

Lo que hace de COM y OLE únicos es que introducen un modelo de programación basado en diseños reutilizables y se han desarrollado proporcionando servicios fundamentales que hacen posible la reutilización del diseño y del código. Como resultado, Microsoft ha ido introduciendo cada vez más tecnologías basadas en OLE (en su modelo de arquitectura de la versión 2.0) incluyendo mejoras a COM (network COM) y tecnologías basadas en OLE incorporadas en el sistema operativo (shell extensions Extensiones del shell).

La arquitectura OLE permite acomodar nuevas tecnologías sin requerir de la participación de Microsoft para ello y sin necesidad de modificar los diseños básicos. OLE ha dejado de considerarse el acrónimo de la vinculación e incrustación de objetos (Object Linking and Embedding), como tecnología específica que servía a propósitos concretos, en su versión 1.0, pasando a ser, en su versión 2.0 el nombre propio de una arquitectura reutilizable para componentes de software, capaz de acomodar nuevos diseños y nuevas tecnologías.

OLE se comprende mejor como el resultado de una curva de crecimiento como la mostrada en la figura. Según ha pasado el tiempo, OLE (basado en COM) se ha expandido para acomodar las nuevas tecnologías, sin llegar a quedarse obsoleta como arquitectura. OLE se puede describir como un conjunto extensible de sistemas con tecnología de objetos, cuya arquitectura es capaz de acomodar diseños nuevos o existentes.



Por Sistemas con tecnologías de objetos queremos referirnos a que es posible trabajar con principios de orientación a objetos, en un sistema operativo en ejecución. Componentes encapsulados, polimórficos y reutilizables existen e interoperan como entidades binarias (en contraposición a las definiciones en código-fuente). Nuevos componentes desarrollados por cualquiera, en cualquier momento, se pueden añadir al sistema en ejecución, extendiendo de ese modo los servicios ofrecidos a las aplicaciones, incluso con dichas aplicaciones ya en ejecución, dando lugar a la denominada arquitectura extensible de servicios.

Un sistema operativo recién instalado ofrece un conjunto de servicios básicos a los desarrolladores, que éstos usarán para crear aplicaciones. COM y OLE aportan la capacidad para que cualquiera amplíe el sistema con nuevos servicios sin necesidad de alterar el sistema operativo, esto es, COM y OLE permiten a cualquiera aportar nuevos servicios que los desarrolladores emplearán en sus nuevas aplicaciones, sin necesidad de instalar ningún tipo de control central o coordinación entre vendedores, constituyendo una infraestructura que permite a cualquiera crear componentes de forma independiente, pero siendo capaz de integrar dichos componentes entre sí de muy variadas maneras.

## 9.2. ¿QUÉ ES UNA APLICACIÓN OLE?

OLE 2.0 no define una plataforma o un sistema operativo (no es posible escribir aplicaciones solo para OLE), de hecho OLE es un medio no un fin.

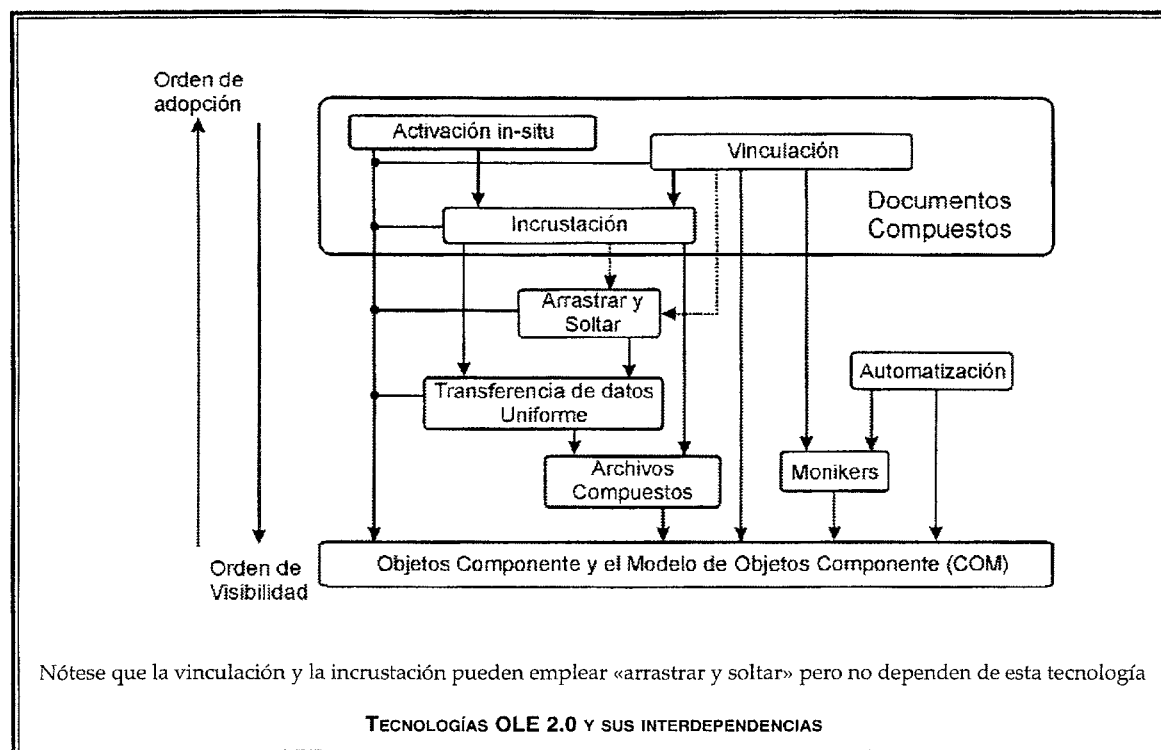
OLE 2.0 es una colección de potentes tecnologías, orientadas a objeto, que uno puede usar para implementar un amplio abanico de funcionalidades en sus propias aplicaciones. No existe una definición única de qué se entiende por una aplicación OLE, cualquier aplicación que emplee alguna de sus tecnologías se puede considerar como tal. OLE es sencillamente una metáfora para todo el conjunto de tecnologías que engloba.

## 9.3. LAS TECNOLOGÍAS OLE.

Las demandas de los usuarios respecto a las aplicaciones, se ajustan a tres categorías: nuevas características; integración entre aplicaciones y consistencia entre aplicaciones. Las diferentes tecnologías que integran el OLE se centran en satisfacer aspectos de una o de varias de las categorías reseñadas, aportando en su conjunto un soporte a todas ellas.

Hay nueve tecnologías OLE principales, como se muestra en la figura. Muchas de las tecnologías ubicadas por encima en el diagrama dependen de las tecnologías mostradas por debajo. Esto significa que, aunque las tecnologías de más alto nivel son más visibles a los clientes, en realidad se deben comenzar a adoptar las últimas, debiendo comenzar de abajo hacia arriba. Hay que tener en cuenta que no es necesario adoptar completamente una tecnología de menor nivel antes de adoptar una superior, en muchos casos será suficiente con conocer cómo trabajan y cómo aplicarlas a las características a desarrollar.





Todas las tecnologías por debajo de los Documentos Compuestos se consideran tecnologías de base. Estas tecnologías comprenden:

- Objetos componente y el modelo de objetos componente: un modelo de programación orientado a objeto adecuado para crear software de sistema reutilizable.
- Archivos compuestos: la implementación de un modelo de almacenamiento estructurado.
- Monikers: son referencias abstractas a objetos.
- Transferencia de datos uniforme: es un potente mecanismo de transferencia de datos.
- Arrastrar y soltar (Drag & Drop): es un potente método para el intercambio de datos.
- Automatización: la capacidad de crear aplicaciones programables.

La colección de tecnologías dentro de los Documentos Compuestos constituyen un estándar de alto nivel para la integración de datos entre aplicaciones. Basta imaginar que hay cosas y lugares donde poner cosas. Las cosas son los objetos del Documento y los lugares son las aplicaciones contenedoras. El cómo se comparten y manipulan dependen de la tecnología específica empleada:

- Incrustación: es el soporte básico de los documentos OLE, donde los objetos se almacenan íntegramente en el propio documento OLE.
- Vinculación: es la capacidad de almacenar solamente un moniker (referencia a un objeto), para un objeto del documento. El moniker señala los datos actuales, almacenados en cualquier lugar. La fuente de datos podría ser el propio objeto incrustado.

- Activación *in-situ*: conocido también como Edición Visual OLE. Normalmente cuando se activan los objetos incrustados o vinculados de un documento OLE, se manipulan en una ventana diferente. La activación *in-situ* mantiene incrustados los objetos en el propio contenedor, accediendo a cualquier herramienta necesaria para su manipulación, dentro del propio contexto del documento. Esto constituye la base de los Controles OLE del futuro.

#### 9.4. VENTAJAS DE LOS DOCUMENTOS OLE.

Las tecnologías que constituyen el OLE aportan características de integración, consistencia y nuevas funcionalidades a cualquier aplicación que haga uso de ellas:

- Integración: cualquier objeto puede incluirse en cualquier contenedor y, cuando se activa *in-situ*, los objetos aparecen como parte del contenedor. En Windows95, incluso el escritorio y la shell del sistema son contenedores y las extensiones del shell se desarrollan como objetos especializados capaces de activarse *in-situ*. Incluso los controles hacen uso de esta tecnología.
- Consistencia: los objetos de la misma clase se comportan igual, independientemente del documento en el que residen. Cada vez que se instala un nuevo servidor sus objetos quedan disponibles inmediatamente para cualquier contenedor, sin cambiar el contenedor. Los monitores permiten efectuar el seguimiento de enlaces rotos o eliminados y con el empleo de conversores y emuladores, los usuarios podrán elegir las herramientas preferidas para trabajar con los objetos.
- Nuevas Características: es posible emplear OLE para integrar cualquier dato desde cualquier fuente como un contenedor, o integrar con cualquier documento como un servidor. Por ejemplo es posible integrar, de manera sencilla, soporte multimedia (sonido e imagen) creando una aplicación contenedora que pueda incorporar este tipo de datos, aunque no sepa nada de multimedia.

#### BIBLIOGRAFÍA

- *Sistemas Operativos Modernos*. Segunda Edición. ANDREW S. TANENBAUM. Ed. Prentice Hall.
- Temario de las pruebas selectivas para el acceso, por promoción interna, al Cuerpo de Gestión de Sistemas e Informática de la Administración del Estado. Ministerio de Administraciones Públicas.
- Apuntes de Sistemas Operativos. Equipo de Sistemas Operativos DISCA/DSIC de la UPV. [www.redes-linux.com/apuntes.php](http://www.redes-linux.com/apuntes.php).
- Microsoft Press y Microsoft Knowledge Base. Documentación oficial de Microsoft.

