



**Universidad Nacional Autónoma de México
Facultad de Ingeniería
Estructuras de datos y algoritmos II**

Tema 5

Archivos

5 Archivos

Objetivo: Interpretar las organizaciones básicas de los archivos, las operaciones que se pueden realizar sobre ellos y su representación mediante diferentes medios de almacenamiento secundario.

5 Archivos

5.1 Generalidades.

5.2 Definición y operaciones.

5.3 Organización de archivos.

5.3.1 Organización lógica.

5.3.2 Organización física.

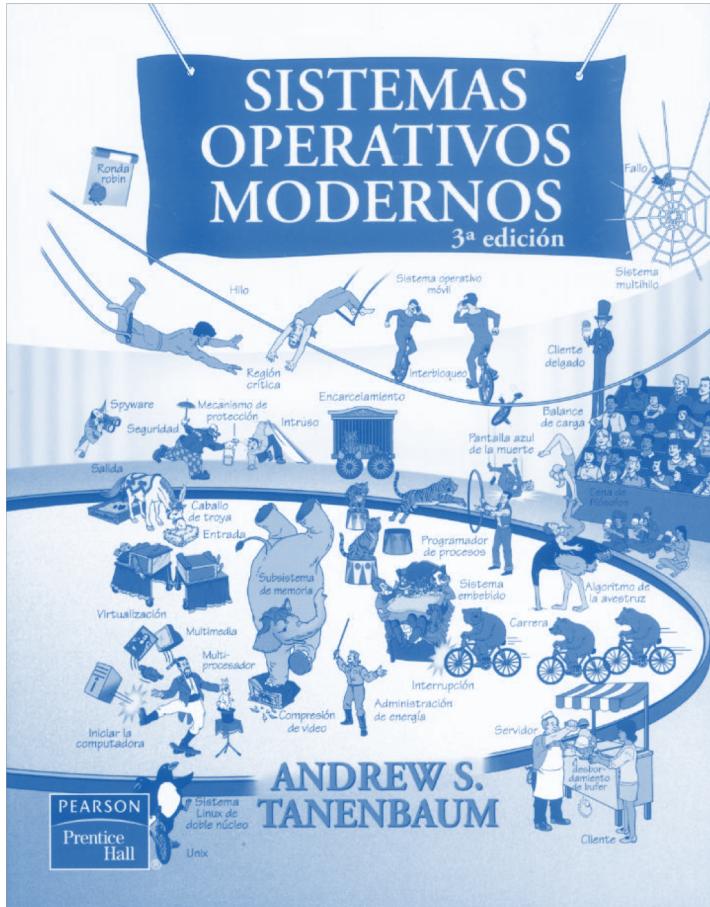
5.4 Acceso a archivos.

5.4.1 Acceso lógico.

5.4.2 Acceso físico.

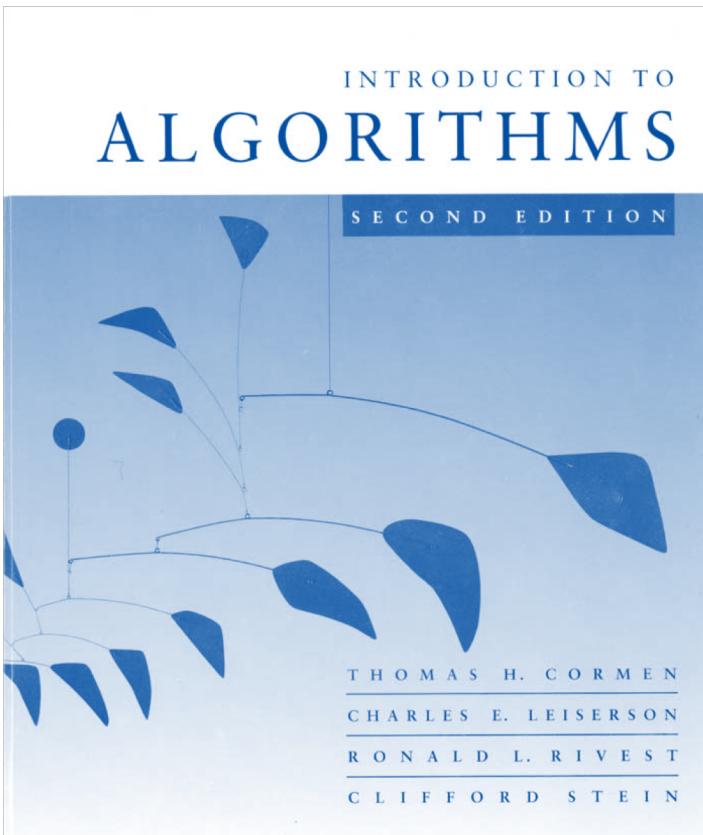
5.5 Sistema de archivos.

Bibliografía



- Sistemas operativos modernos. Andrew S. Tanenbaum, Prentice Hall, 3ra edición.

Bibliografía



- Introduction to Algorithms.
Thomas H. Cormen,
Charles E. Leiserson,
Ronald L. Rivest, Clifford
Stein, McGraw-Hill.

Archivos

- Tema 4

La palabra archivo procede del latín **archivum**, y se utiliza para nombrar al conjunto ordenado de documentos que una sociedad, una institución o una persona elabora en el marco de sus actividades y funciones.

Dentro de una computadora, un archivo se puede definir como un conjunto lógico de información o de datos que se almacena con un nombre y se configura como una unidad autónoma completa para el sistema de archivos y para el usuario.

5.1

Generalidades

5. Archivos

Durante la ejecución de un programa, el almacenamiento de datos se realiza en memoria principal, por lo tanto, la información es volátil (por la naturaleza propia de este tipo de dispositivo de almacenamiento) y, en cuanto se termina de ejecutar el programa, los datos almacenados (computados) se pierden.

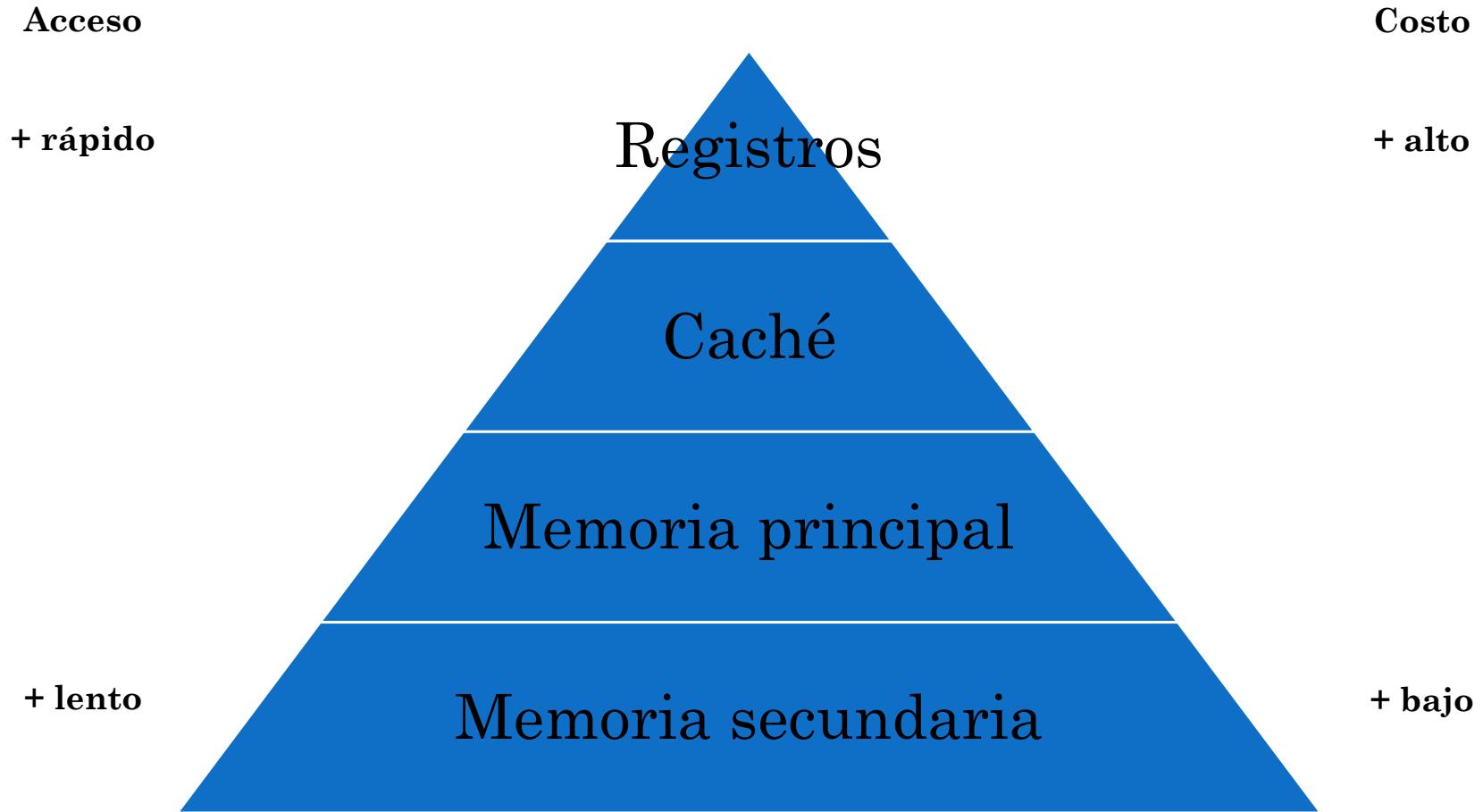
Para conservar la información generada durante la ejecución se utilizan los archivos, los cuales hacen uso de la memoria secundaria.

Los archivos permiten mantener la información organizada por un largo período de tiempo en medios de almacenamiento secundario (y masivo), de tal manera que solo cuando se necesita dicha información se traslada a la memoria principal.

La transmisión de datos de la memoria secundaria a la memoria principal y viceversa se realiza a través de un buffer de datos.



A continuación se muestran las memorias de una computadora:



A continuación se muestran una tabla que compara el costo con el tamaño de las memorias de una computadora.

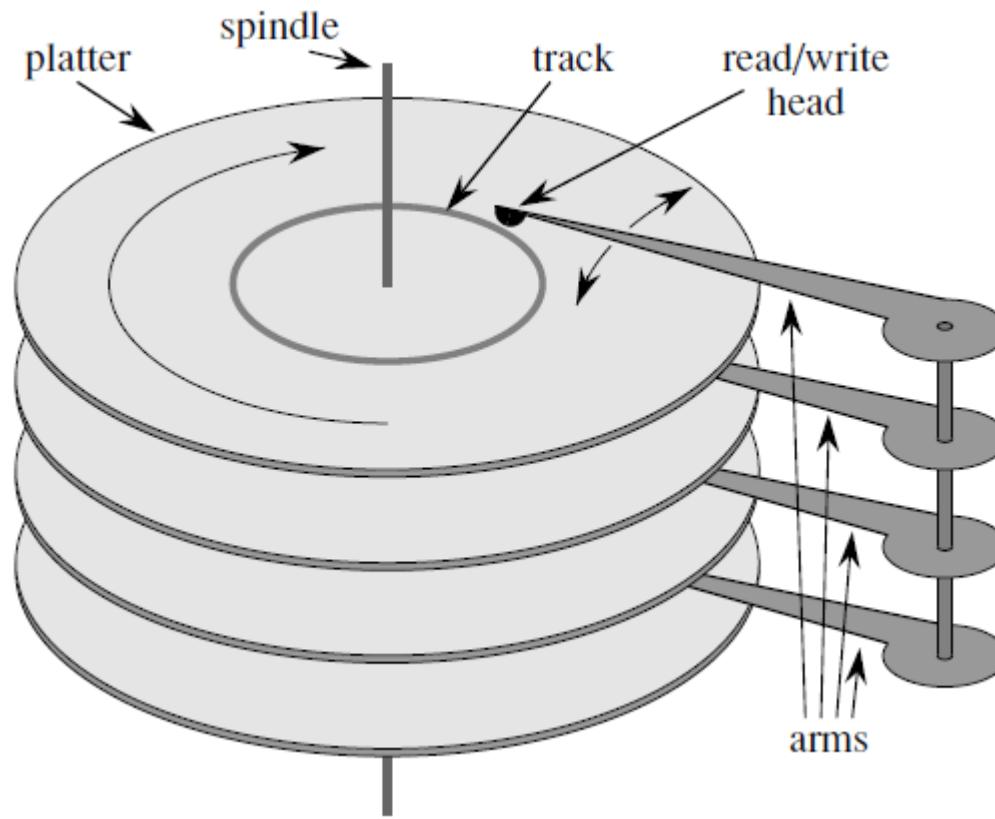
Tipo de memoria	Tiempo de acceso	Tamaño típico
Registros	1 [ns]	1 [kb]
Caché	5 - 20 [ns]	1 [Mb]
Memoria principal	60-80 [ns]	4 [Gb]
Memoria secundaria	10 [ns]	1 [Tb]

Disco duro

Una unidad de disco duro está formada por varios platos (discos) que giran a velocidad constante sobre un eje común. La superficie de cada plato está elaborada de compuestos de vidrio, cerámica o aluminio, finamente pulidos y revestidos por ambos lados con una capa muy delgada de una aleación metálica.

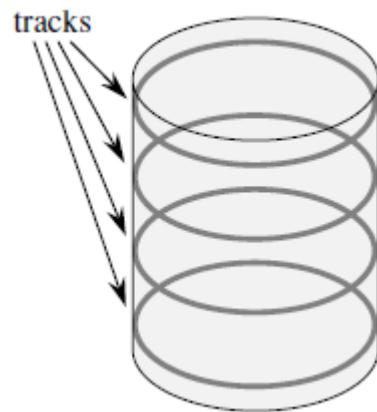
Los platos son leídos o escritos a través de una cabeza que se encuentra al final de un brazo. Los brazos están físicamente acoplados y pueden acercar o alejar las cabezas del eje.

A continuación se muestra de manera gráfica las partes de un disco duro



Cuando una determinada cabeza está quieta, la superficie que se encuentra debajo se llama pista.

Las cabezas de lectura / escritura se encuentran alineadas verticalmente en todo momento y, por tanto, el conjunto de pistas bajo cada una puede ser accedida simultáneamente. Al conjunto de pistas se le conoce como cilindro.



A pesar de que un disco duro es más barato y tiene más capacidad de almacenamiento que la memoria principal, es mucho más lento debido a sus partes mecánicas.

Los componentes mecánicos de un disco son dos, la rotación de los platos y el desplazamiento del brazo. La velocidad de rotación de un disco oscila entre las 5400 y las 15000, siendo la más común 7200 RPM.

Una rotación toma 8.33 milisegundos (aproximadamente), mientras que el acceso a la memoria principal es alrededor de 100 nanosegundos. En otras palabras, si se requiere esperar una rotación completa para que un elemento en particular se sitúe debajo de la cabeza lectora, se podría acceder casi 100,000 veces a la memoria primaria durante esa espera.

Árbol B

Para amortizar el tiempo gastado en la espera de movimientos mecánicos, se debe acceder al disco para obtener varios elementos a la vez. La información se divide en un número de páginas de bits del mismo tamaño, las cuales aparecen contiguas en los cilindros del disco.

El tamaño típico de una página en un disco está en el orden de entre 2^{11} y 2^{14} bytes de longitud. Una vez que la cabeza lectora se posiciona en la ubicación deseada, la lectura o escritura en un disco magnético es completamente electrónica (rápida) y, por tanto, se pueden leer / escribir gran cantidad de datos de manera rápida.

A menudo, toma más tiempo acceder a una página del disco para leer información de lo que le toma a una computadora para examinar toda la información leída.

El número de accesos a disco es una medida que está en términos del número de páginas que se requieren leer o escribir. Además, el acceso al disco no es un tiempo constante, ya que depende de la pista donde se encuentre la página.

La cantidad de información que puede manejar un árbol B es tan grande que toda no cabe en la memoria principal. Por tanto, solo se copian las páginas seleccionadas a la memoria principal según se requieran, y se van escribiendo en el disco las páginas que han cambiado.

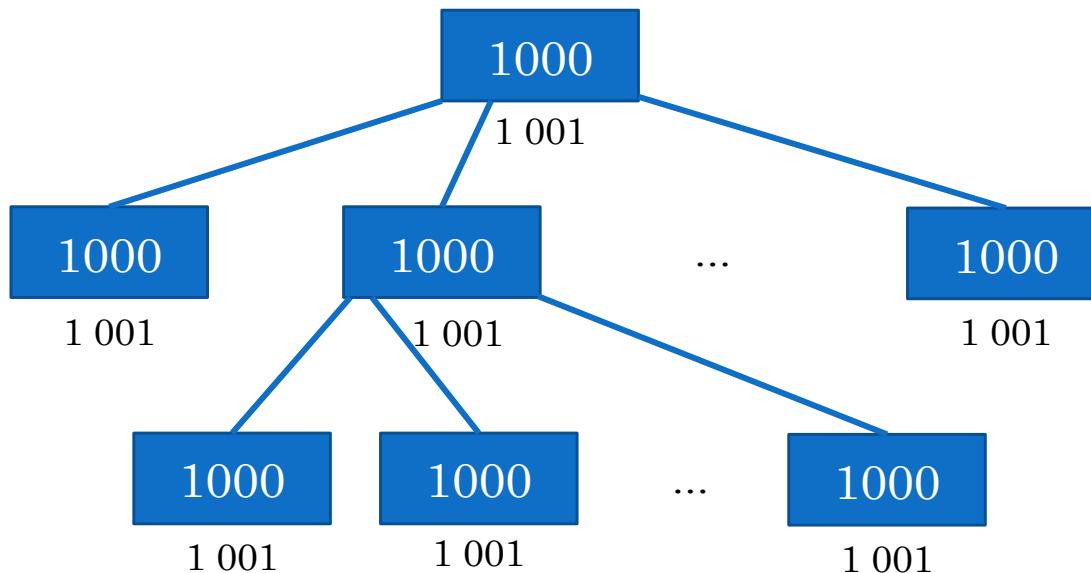
Los algoritmos de los árboles B están diseñados para que sólo un número constante de páginas estén en la memoria principal a la vez.

Un árbol B de altura 2 contiene más de un billón de llaves:

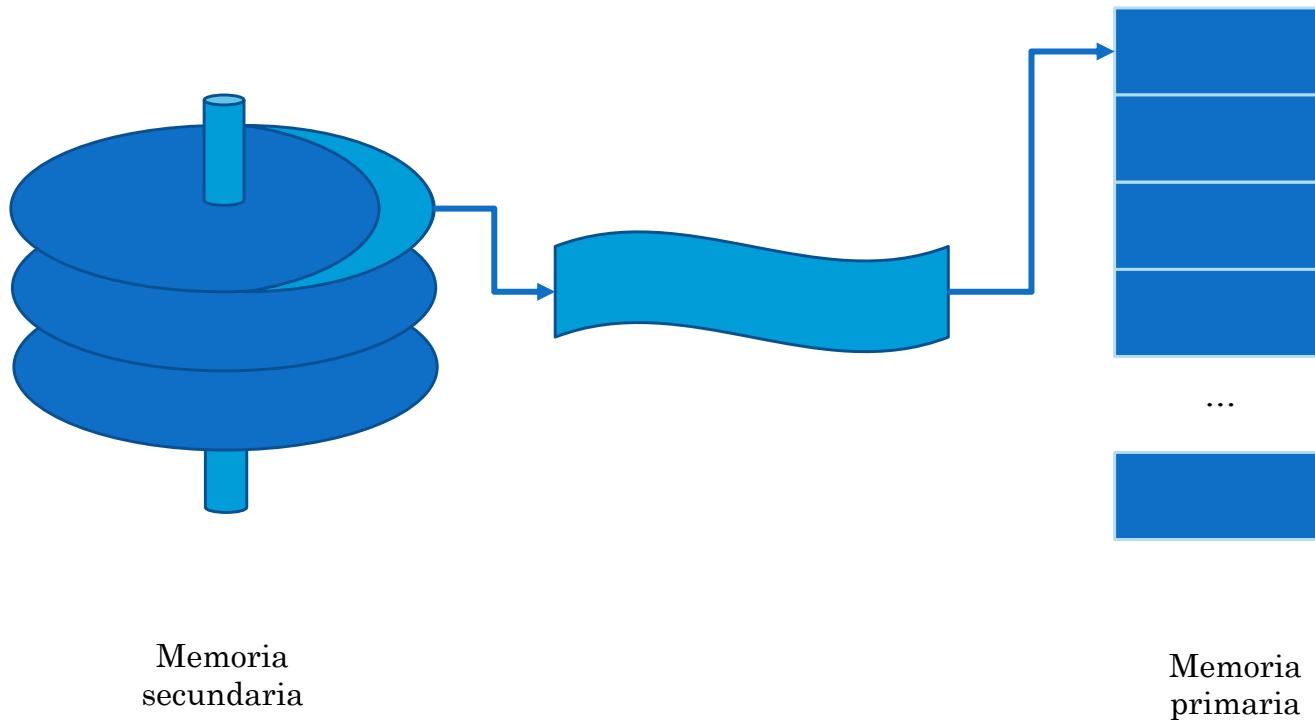
1 nodo:
1 000 llaves

1 001 nodos:
1 001 000 llaves

1 002 001 nodos:
1 002 001 000 llaves



En este tema se va a abordar el proceso de almacenamiento o recuperación de información entre un medio de almacenamiento primario y uno secundario.



5.2

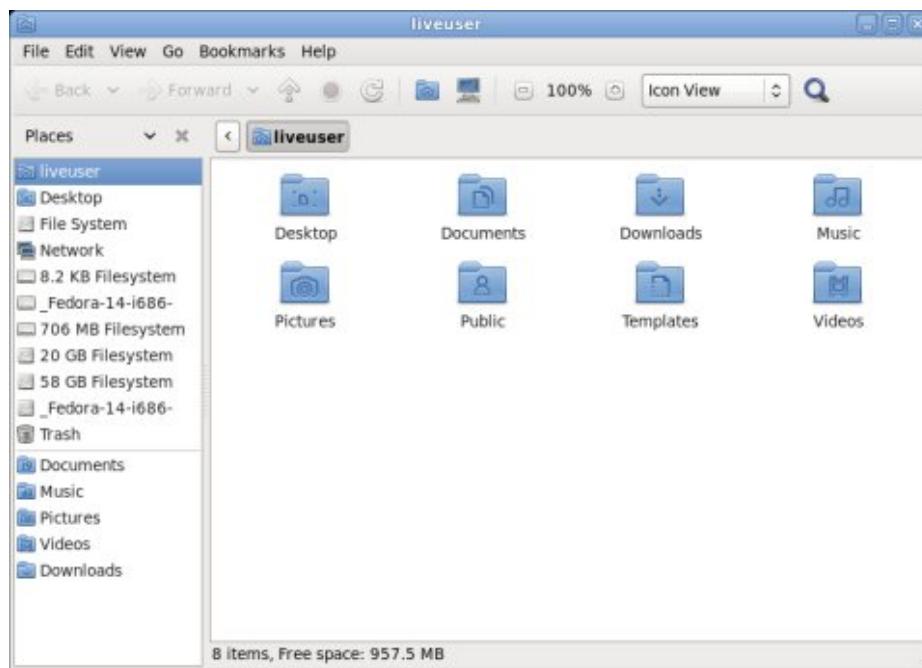
Definiciones y operaciones

5. Archivos

Un archivo es la unidad básica de almacenamiento de información a largo plazo. Está constituido por un conjunto de elementos de información (registros) que poseen un significado para el usuario.

Para su estudio, los archivos pueden dividirse en dos niveles: lógico y físico.

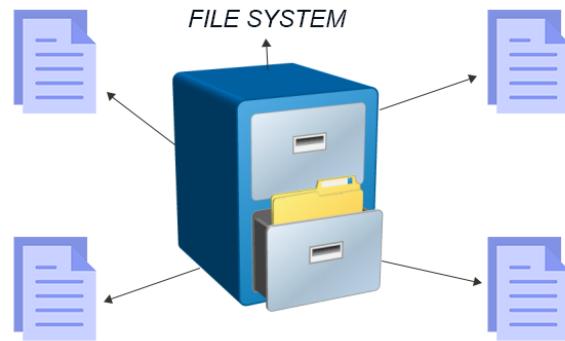
El nivel lógico hace referencia a la forma en la que el usuario ve el archivo, es decir, contiene la información con la que trabaja el usuario.



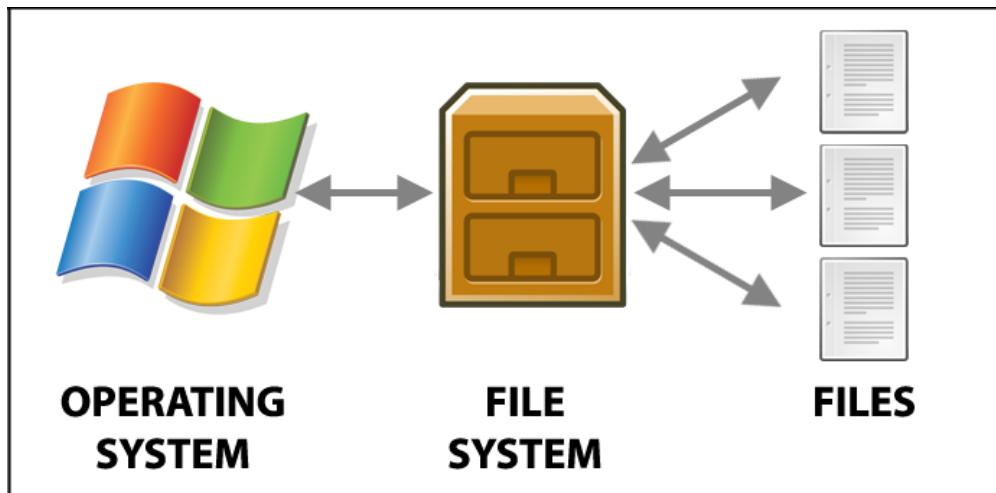
El nivel físico se refiere a la manera en la que el archivo se encuentra almacenado en la memoria secundaria. Almacena la cantidad de datos que puede transferirse en una operación de I/O a través del buffer.



Es posible establecer diferentes organizaciones de archivos, así como diferentes métodos de acceso para ambos niveles. Por tanto, es necesario contar con alguna interfaz que permita manipular los archivos de manera independiente a la forma en la que se almacenan en la memoria secundaria.



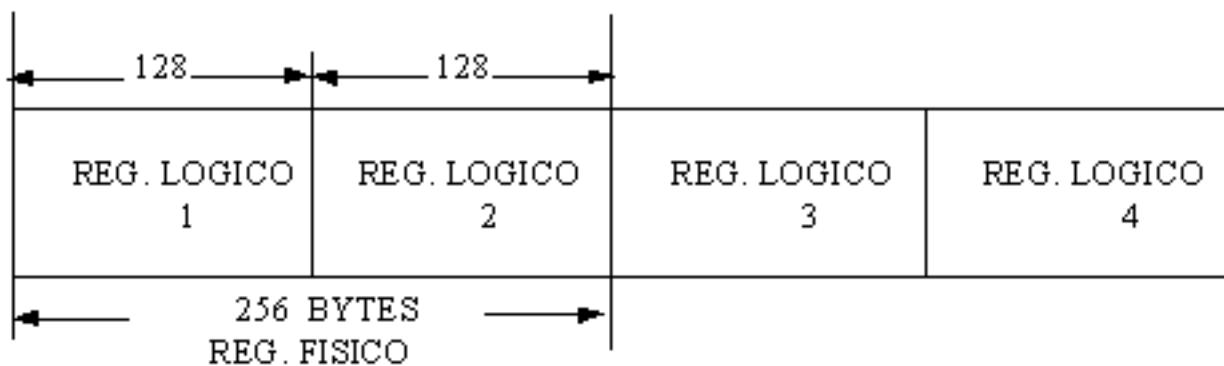
El sistema de archivos, el cual forma parte del sistema operativo, permite establecer una correspondencia entre los archivos lógicos y los archivos físicos.



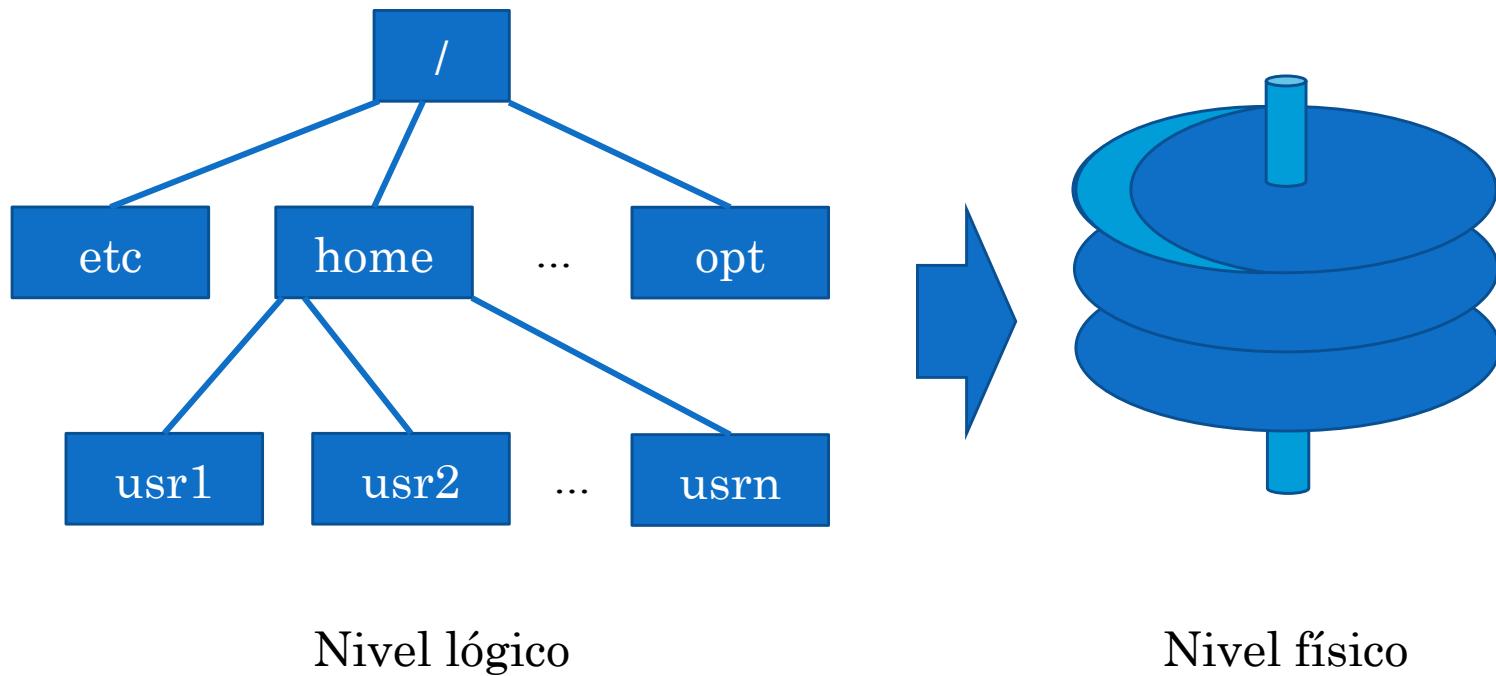
Las principales operaciones que se pueden realizar sobre un archivo son:

- Crear: guardar un archivo en un dispositivo de almacenamiento secundario.
- Leer: se transmite la información del archivo desde la memoria secundaria hacia la memoria principal.
- Escribir (actualizar): se transmite la información hacia el archivo desde la memoria principal hacia la memoria secundaria.
- Borrar: se borra la entrada del directorio del usuario.

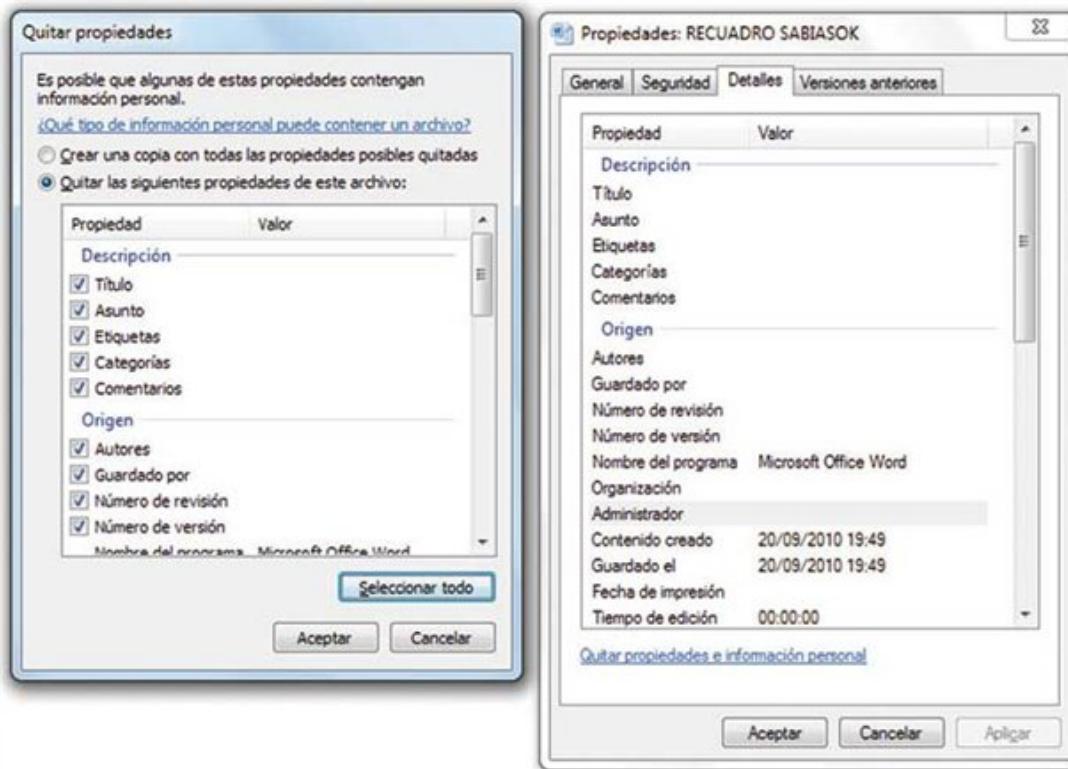
La transmisión de información (tanto de lectura como de escritura) se lleva a cabo en unidades de tamaño fijo llamadas bloques. El tamaño de los bloques, a menudo, corresponde al tamaño de los registros físicos. Además, es común que en los bloques se almacenen varios registros lógicos.



El sistema de archivos permite establecer una correspondencia entre los archivos lógicos y los archivos físicos.



Todo archivo tiene un nombre y sus datos. Además, los sistemas operativos asocian información adicional a cada archivo, por ejemplo, la fecha y hora de la última modificación del archivo o su tamaño. A la información adicional se le conoce como atributos del archivo (conocidos también como metadatos).



5.3

Organización de archivos

5. Archivos

La organización de archivos define la forma en la que los registros se disponen sobre el soporte de almacenamiento.

Por tanto, se deben analizar tanto la organización física (en un dispositivo de almacenamiento) como la organización lógica (disponible para el usuario),

Ejemplo

Se desea almacenar la siguiente información en la memoria secundaria del equipo:

jossie	peterburg
password	hola1234
José Luis	Pedro Díaz

Ejemplo

Para ahorrar espacio se almacena la información del archivo de forma consecutiva:

```
jossiepasswordJosé Luispeterburghola1234Pedro Díaz
```

El problema ahora es recuperar la información, ¿cómo saber qué campos corresponde a cada uno de los registros? Es aquí donde la organización lógica tiene su razón de ser.

Ejemplo

El problema anterior se debe resolver al crear el archivo, proporcionando un método fácil para recuperar la información. Las soluciones más obvias son:

- Guardar los campos con una longitud fija.
- Colocar un delimitador al final de cada campo.
- Crear un diccionario (llave=valor) para identificar cada campo.

Estas soluciones depende de la manera en la que se organicen los archivos dentro de un sistema de archivos.

5.3.1 Organización lógica.

Un archivo está formado por un conjunto de registros lógicos. Este conjunto puede estar formado por registros con llave o sin llave. A su vez, un registro puede ser de longitud fija o de longitud variable.

Los archivos que utilizan registros sin llave están constituidos por una secuencia de elementos lógicos. Se pueden clasificar en:

- Archivos sin llave de longitud fija: archivos con registros del mismo tamaño.
- Archivos sin llave de longitud variable: archivos con registros de diferente tamaño.



Ejemplo

Un registro sin llave de longitud fija define un tamaño máximo por cada campo (dependiendo del valor más largo que se desee almacenar).

jossie
password
José Luis
peterburg
hola1234
Pedro Díaz

Ventajas:

- Para encontrar un valor en una posición determinada, sólo se salta el número de campos deseados y se llega al valor.

Desventajas:

- Si los campos varían mucho en tamaño, el archivo se puede hacer demasiado grande.
- Cuando se inserta un registro nuevo más grande que el tamaño reservado, se deben redimensionar el tamaño de todos los campos.

Ejemplo

Un registro sin llave de longitud variable define un delimitador al final de cada campo.

Jossie	password	José Luis
peterburg	hola1234	Pedro Díaz

Ventajas:

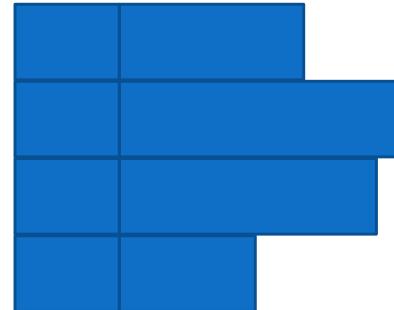
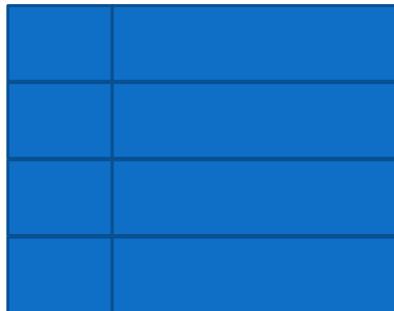
- Se almacena solo el tamaño que ocupe el campo más uno (el delimitador) lo que ahorra espacio en el disco.

Desventajas:

- La selección de un delimitador es de vital importancia, ya que si el delimitador elegido coincide con un carácter del campo, puede haber pérdida o inconsistencia en la información.

Los archivos que utilizan registros con llave requieren de manera explícita un identificador por cada campo. Se pueden clasificar en:

- Archivos con llave de longitud fija: archivos con registros del mismo tamaño.
- Archivos con llave de longitud variable: archivos con registros de diferente tamaño.



Ejemplo

Un registro con llave de longitud fija define una llave para cada registro, así como un tamaño máximo por cada campo (dependiendo del valor más largo que se desee almacenar).



Ventajas:

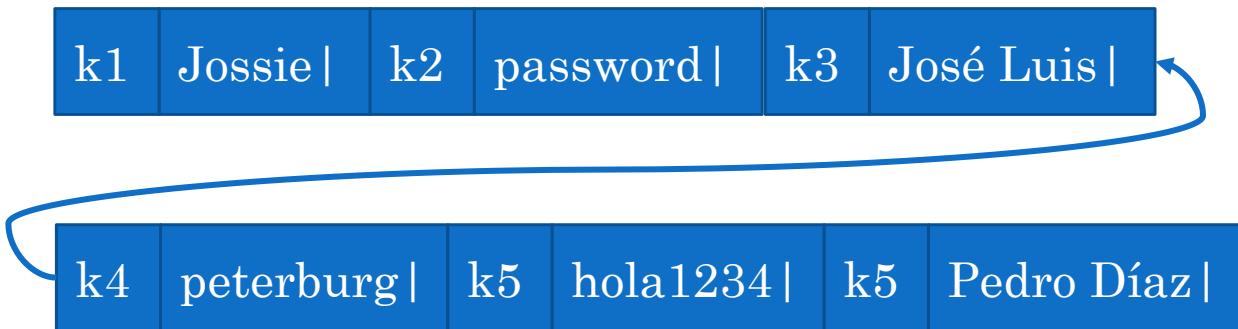
- El acceso a un determinado valor es directo gracias al uso de llaves.

Desventajas:

- Si los campos varían mucho en tamaño, el archivo se puede hacer demasiado grande.
- Cuando se inserta un registro nuevo más grande que el tamaño reservado, se deben redimensionar el tamaño de todos los campos.

Ejemplo

Un registro con llave de longitud variable define un delimitador al final de cada campo.



Ventajas:

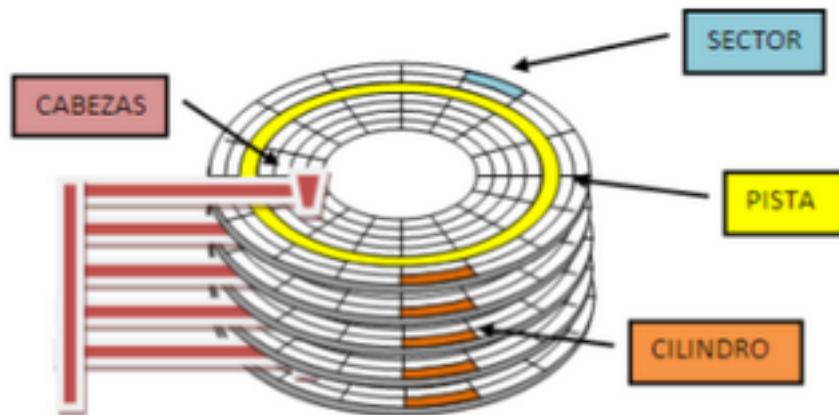
- Se almacena la llave, el tamaño que ocupa el campo y un espacio más para el delimitador.

Desventajas:

- La selección de un delimitador es de vital importancia, ya que si el delimitador elegido coincide con un carácter del campo, puede haber pérdida o inconsistencia en la información.

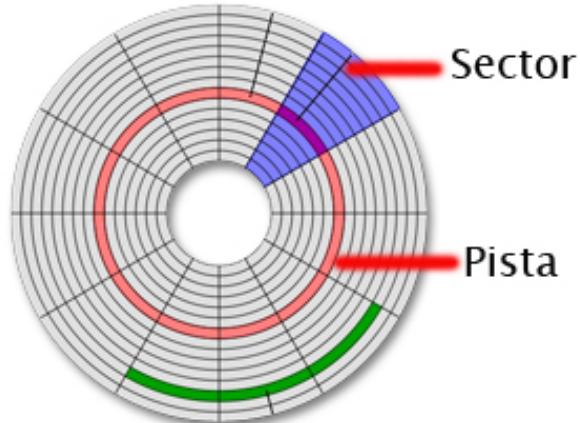
5.3.2 Organización física.

La organización física se refiere a la manera en la que la información es almacenada en la memoria secundaria. En la organización física los registros pueden ser de tamaño fijo o de tamaño variable.



Un archivo físico en un disco duro es una colección de registros físicos que pueden estar organizados de tres formas:

- Organización contigua
- Organización ligada
- Organización en tabla de mapeo



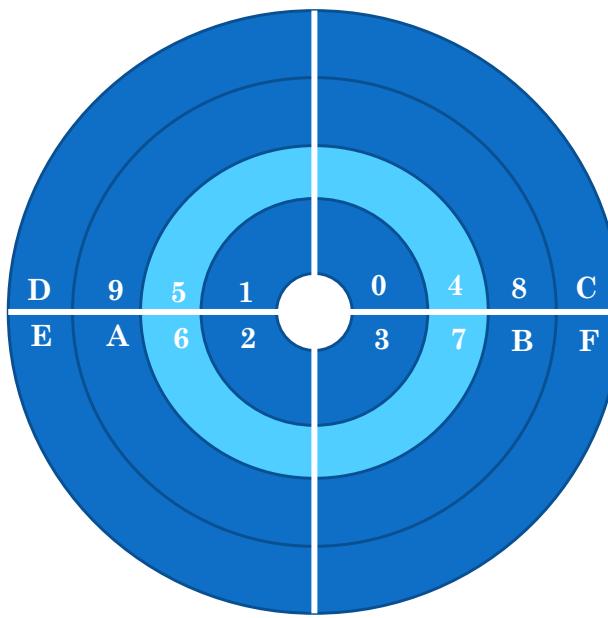
Organización contigua

En este tipo de organizaciones, el archivo se almacena en registros físicos aledaños, siguiendo una secuencia normal de direcciones (por ejemplo, registros sobre la misma pista).

Para saber qué registros físicos forman el archivo, se debe conocer la dirección del primer registro y el número de registros que forman el archivo.

Ejemplo

Dado un disco con cuatro pistas y cuatro sectores por pista, se almacena un archivo en los sectores 4, 5, 6 y 7 del disco.



En este caso, el archivo inicia en el sector 4 y ocupa 4 registros en el disco (hasta el sector 7).

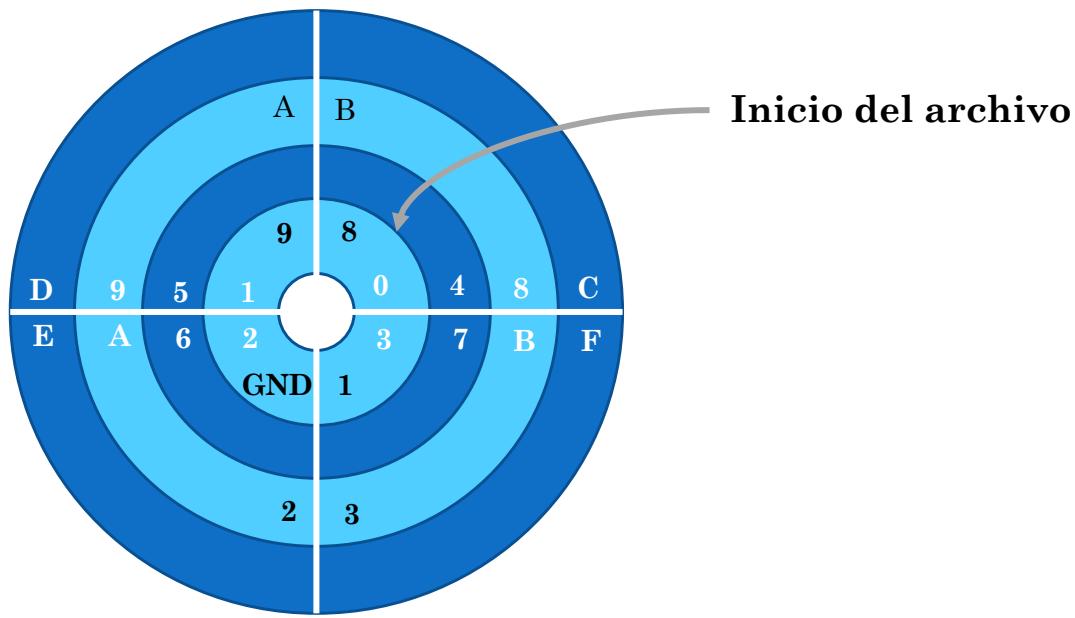
Organización ligada

En este tipo de organizaciones, el archivo se almacena en registros físicos y un campo extra que se utiliza para indicar la dirección del siguiente registro.

Para saber qué registros físicos forman el archivo, se debe conocer la dirección del primer registro, únicamente.

Ejemplo

Dado un disco con cuatro pistas y cuatro sectores por pista, se almacena un archivo ligado a partir del sector 0.



En este caso, el archivo inicia en el sector 0 y de ahí se buscan todos los registros que componen el archivo con la liga de cada uno.

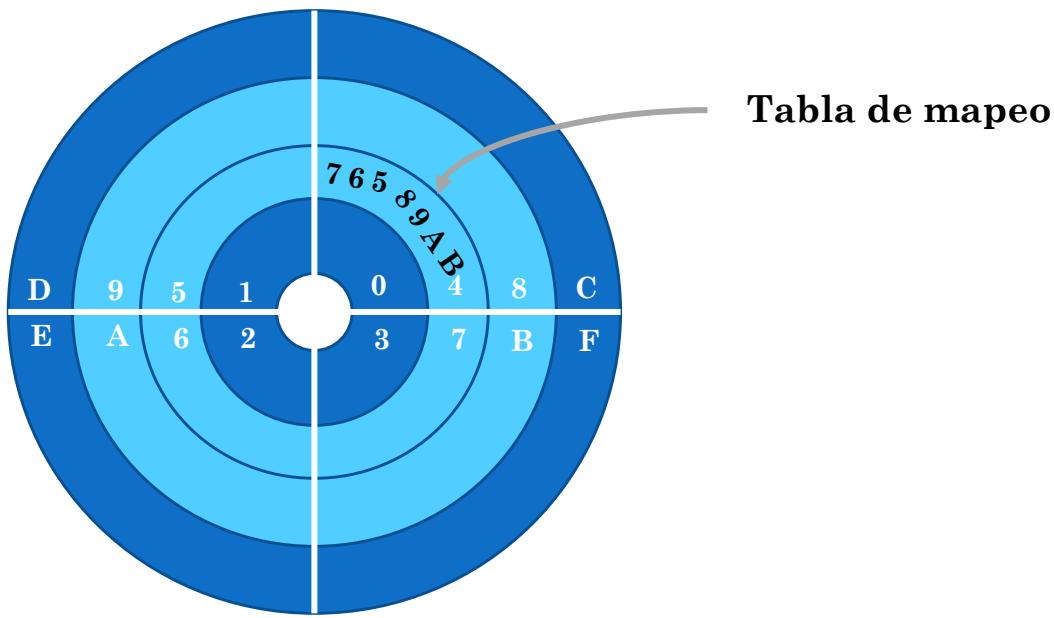
Organización por tabla de mapeo

Consiste en una lista de apuntadores hacia los registros físicos que forman el archivo.

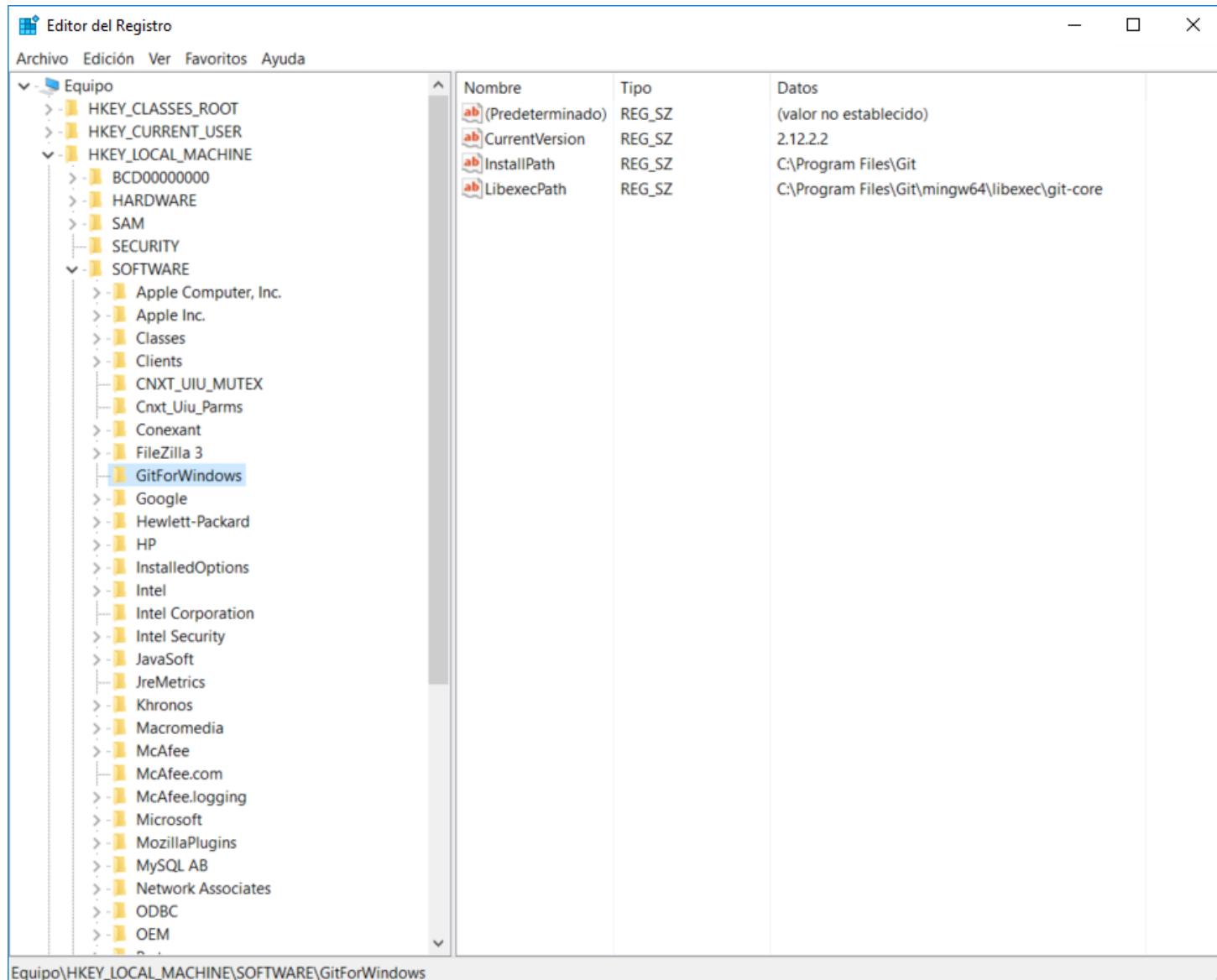
Todas las ligas de los registros se extraen y se almacenan en una tabla de mapeo. Para conocer los registros que forman al archivo, solo se necesita un apuntador al primer registro de la tabla.

Ejemplo

Dado un disco con cuatro pistas y cuatro sectores por pista, se almacena un archivo por tabla de mapeo en el sector 4.



En este caso, solo se requiere el apuntador al registro que contiene la tabla de mapeo.



5.4

Acceso a archivos

5. Archivos

El acceso a archivo se refiere a la manera de llegar al nodo deseado para leer o escribir información.

Debido a que se poseen dos tipos de organizaciones, también se poseen dos tipos de acceso a archivos, el acceso lógico y el acceso físico.

5.4.1 Acceso lógico

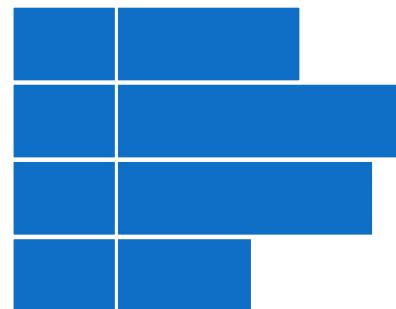
A cada registro lógico que forma el archivo se le asigna un número consecutivo de acuerdo a la posición relativa que ocupan con respecto al inicio del archivo.

El acceso a los registros que van a ser procesados se puede llevar a cabo utilizando dos métodos: acceso secuencial y acceso directo.

Acceso lógico secuencial

Consiste en obtener los recursos de forma consecutiva utilizando el número asociado a cada uno de ellos.

El acceso secuencial se puede llevar a cabo sobre archivos con registros sin llave o con llave, de longitud fija o de longitud variable.



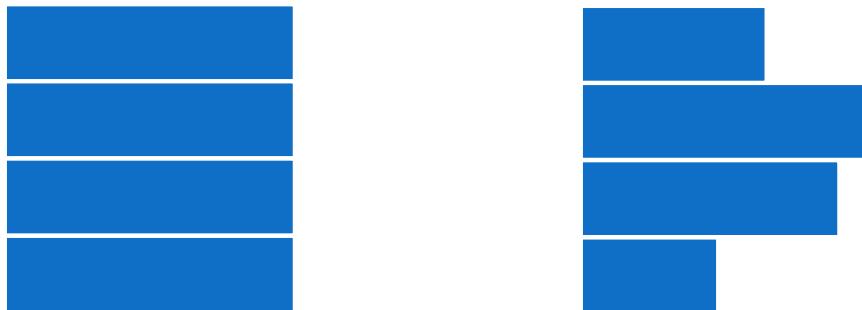
Procesar registros de un archivo con un tipo de acceso secuencial puede ser lento, por tanto, se recomienda utilizar este acceso cuando se va a realizar la misma operación sobre todos los registros del archivo y no se requiere de un tiempo de respuesta breve.

Debido a que los registros se acceden de forma consecutiva, se respeta el orden de aparición de los mismos para formar el archivo. La complejidad de este tipo de acceso es de $O(n)$.

Acceso lógico directo

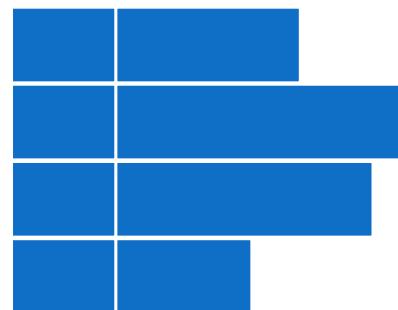
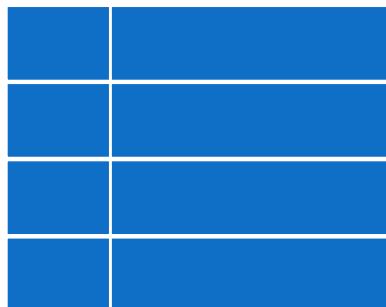
El acceso directo se puede llevar a cabo en cualquier orden, esto es, se puede obtener un registro sin importar cuál ha sido procesado o cuál se procesará después.

El acceso a los archivos formados por registros sin llave (tanto de longitud fija como de longitud variable) se realiza utilizando el número de registro como índice.



El acceso a los archivos formados por registros con llave (tanto de longitud fija como de longitud variable) se realiza utilizando la llave del registro como índice.

El tiempo de acceso a los registros de un archivo de manera directa es muy rápido, $O(1)$, independientemente del número de registros que se vayan a procesar.

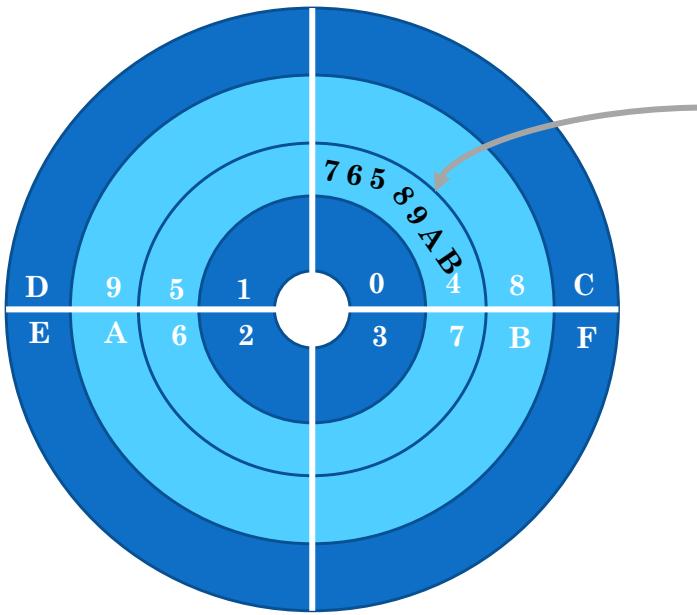
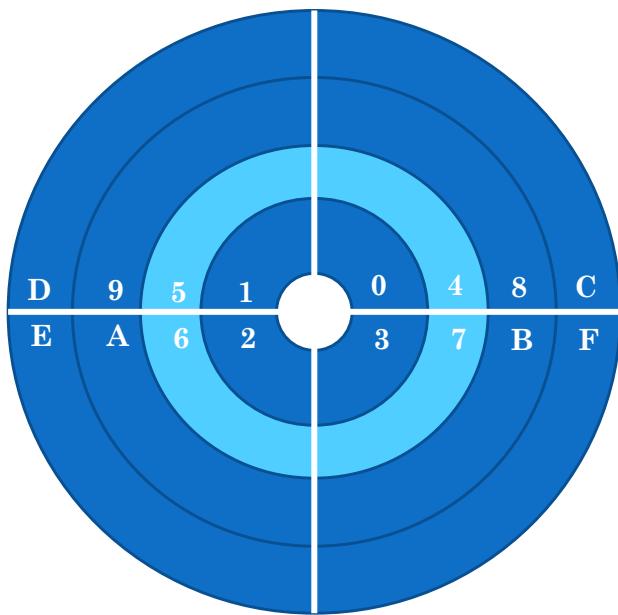


5.4.2 Acceso físico

El acceso a los registros de un archivo físico también se puede realizar de manera secuencial o de manera directa. La diferencia con los accesos lógicos radica en que el acceso físico usa las direcciones físicas de los registros, mientras que el acceso lógico usa las direcciones relativas.

Dentro de un disco es posible acceder a registros físicos de dos maneras distintas:

- Acceso secuencial físico
- Acceso directo físico



Acceso secuencial físico

El acceso secuencial físico consiste en obtener los registros físicos de acuerdo a su posición relativa a la dirección de inicio en el archivo.

El modo de acceso secuencial se puede llevar a cabo en cualquier tipo de organización, es decir, ya sea contigua, ligada o por tabla de mapeo.

Acceso secuencial físico contiguo

El acceso secuencial físico en una organización contigua se lleva a cabo utilizando la dirección del primer registro físico del archivo, para así procesar los registros en la secuencia de direcciones aledañas hasta llegar al número de registros que conforman el archivo.

Acceso secuencial físico ligado

El acceso secuencial físico en una organización ligada se lleva a cabo utilizando la dirección del primer registro físico del archivo y siguiendo la secuencia indicada por las direcciones guardadas en el campo de ligas de los registros que forman el sistema.

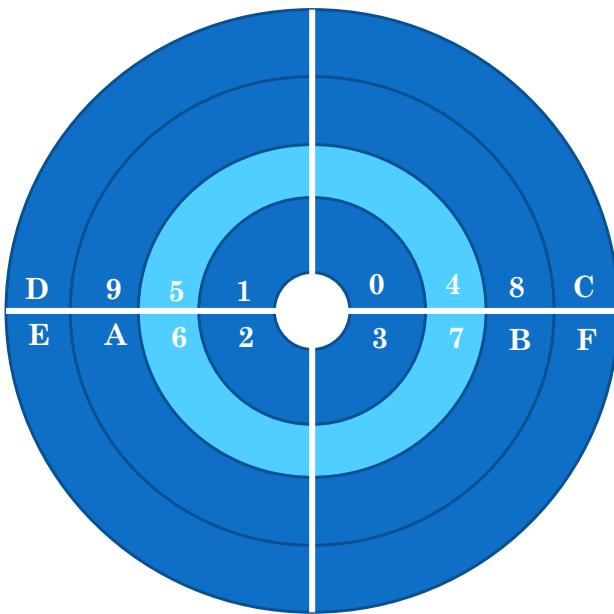
Acceso secuencial físico por tabla de mapeo

El acceso secuencial físico en una organización por tabla de mapeo se lleva a cabo utilizando la dirección de inicio de la tabla para recorrerla y procesar así los registros en la secuencia en la que aparecen y que forman al archivo.

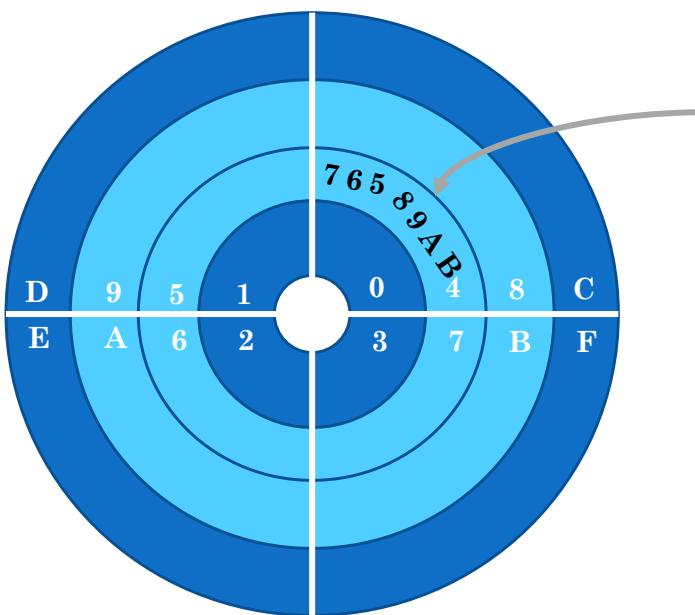
Acceso directo físico

El acceso directo permite obtener cualquier registro del archivo sin importar el orden de procesamiento. Este tipo de acceso solo se puede realizar en archivos organizados en forma contigua de tamaño fijo o en tabla de mapeo.

En la organización contigua de longitud física el acceso se lleva a cabo utilizando el número físico del registro como índice, así como un desplazamiento.



En la organización con tabla de mapeo es necesario realizar dos accesos, el primero a la tabla de mapeo para obtener la dirección física del registro y el segundo para obtener el registro especificado.



5.5 Sistema de archivos

5. Archivos

El sistema de archivos de una computadora se encarga de gestionar la manera en la que los archivos son nombrados, así como el lugar donde son guardados de manera lógica para poder guardar y recuperar la información.

Por ejemplo, los sistemas de archivos que los sistemas operativos basados en DOS, Windows, Mac o Unix son todos de tipo jerárquico, los archivos se sitúan en carpetas y subcarpetas en una estructura de árbol.

Los sistemas de archivos (file systems) especifican sus propias convenciones para nombrar archivos, como son el máximo número de caracteres en el nombre, los caracteres que se pueden utilizar, la longitud de la extensión, así como un formato para especificar la ruta donde se almacenará el archivo dentro de la estructura de directorios.

Los sistemas de archivos más comunes que se pueden encontrar son:

FAT32: Es un tipo de sistemas de archivos antiguo de Windows, pero bastante utilizado actualmente es dispositivos de almacenamiento extraibles y de poca capacidad. Es utilizado para tener compatibilidad con otros dispositivos de poco almacenamiento como cámaras digitales, consolas o dispositivos que solo soportan ese sistema.

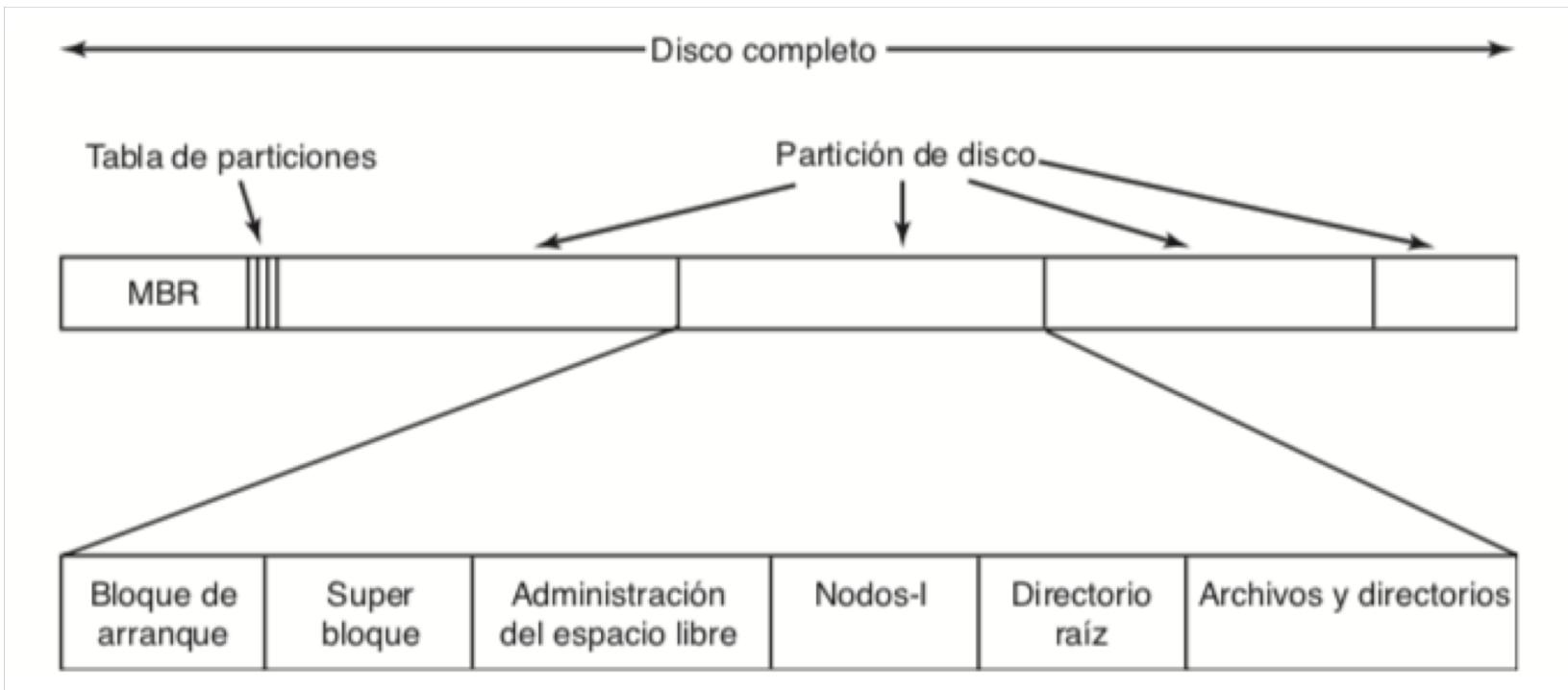
NTFS: Es el sistema de archivos más reciente de Windows, a partir de Windows XP. Reconoce FAT32.

HFS+: Mac utiliza HFS+ para particiones internas. En particiones externas con Time Machine también es utilizado este formato. Las Mac también pueden reconocer formatos FAT32 para lectura y escritura y NTFS para lectura.

Ext2/Ext3/Ext4: Son los tipos de archivos GNU/Linux por excelencia. Ext2 es el formato utilizado en las primeras versiones del sistema. Ext3 crea un sistema más robusto, permite recuperarse ante apagones imprevistos. Ext4 es más moderno y rápido. Windows y Mac no soportan este tipo de sistemas (por defecto). GNU/Linux puede leer y escribir tanto en FAT32 como en NTFS.

Btrfs: Better file system es un sistema de archivos nuevo en GNU/Linux. El objetivo de este sistema es brindar características adicionales que permitan al SO escalar a grandes cantidades de almacenamiento.

Swap: En Linux, swap no es en realidad un sistema de archivos, es más bien un espacio que el SO puede utilizar cuando éste se queda sin memoria física. Es análogo al page file de Windows, pero requiere una partición dedicada.



Distribución de un sistema de archivos

Cuando se realiza una solicitud de acceso a un archivo se debe indicar la operación a realizar (lectura o escritura), el nombre del archivo, el número de registro lógico sobre el que se efectuará la operación y una dirección de memoria de donde la información será leída (recuperación) o en donde la información será escrita (almacenamiento).

El sistema de archivos tiene como funciones principales transformar una solicitud de acceso a un archivo lógico en un acceso a un archivo físico.

El sistema de archivos cuenta con diferentes módulos que permiten transformar las solicitudes de acceso. Entre los más importantes están:

- El sistema de archivos simbólicos
- El sistema de archivos básicos
- El sistema de archivos lógicos
- El sistema de archivos físicos
- El sistema de manejo del dispositivo.

Sistema de archivos simbólicos

La función de este sistema consiste en transformar el nombre simbólico del archivo en un identificador interno, mediante el uso de un directorio de archivos simbólicos donde aparecen los nombres simbólicos de los archivos (en la primera columna), así como los identificadores correspondientes a cada uno de ellos (en la segunda columna).

Nombre del archivo	Identificador
---------------------------	----------------------

Sistema de archivos básicos

Este sistema utiliza el identificador asociado a cada archivo como un índice para dar acceso al directorio de archivos básicos donde se encuentra la información acerca del tamaño de registro lógico, el número de registros lógicos que forman el archivo y la dirección de inicio del primer registro físico.

Cuando un archivo está en uso, la información del archivo se copia a la memoria principal y permanece ahí durante el tiempo de procesamiento.

Identificador interno	Tamaño del registro lógico	Registros lógicos que forman el archivo	Dirección de inicio del primer registro físico
------------------------------	-----------------------------------	--	---

Sistema de archivos lógicos

Este sistema se encarga de transformar la solicitud de acceso a un registro lógico en una solicitud equivalente para acceder a una cadena de bytes. Para ello se requiere obtener la dirección lógica de la cadena así como su longitud. Estos datos se pueden obtener utilizando el número de registro lógico y la longitud del mismo.

DLCB

TCB

Sistema de archivos físicos

Este sistema se encarga de convertir la dirección lógica de la cadena y su longitud en un número físico de registro y en un desplazamiento dentro del mismo.

NRF

Desplazamiento

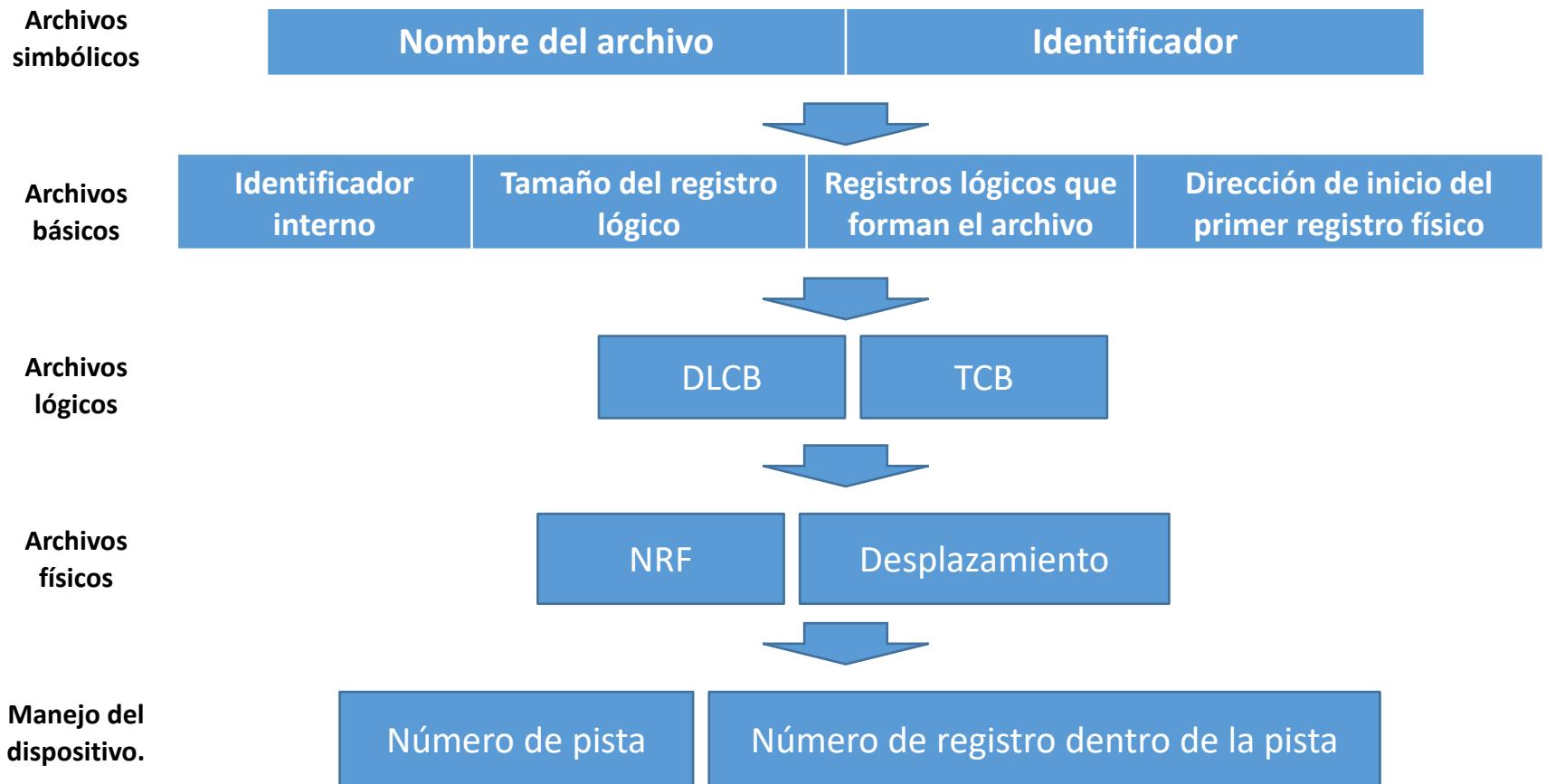
Sistema de manejo del dispositivo

Este módulo se encarga de transformar el número de bloque físico en una dirección física, de acuerdo al formato requerido por el dispositivo, además, realiza la operación solicitada.

Num
Pista

Num Reg
Dentro Pista

Sistema de archivos



Ejemplo

Desde una aplicación se realiza la petición de apertura de un archivo:

```
fopen("prueba", "r")
```

Ejemplo

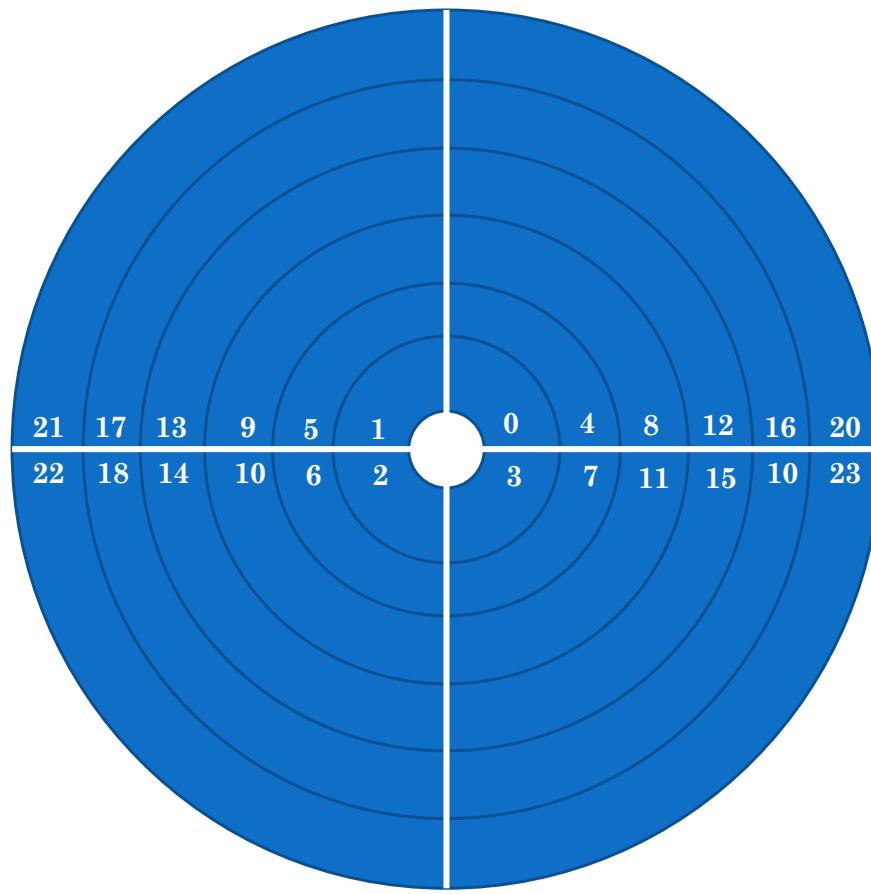
Físicamente, el sistema de archivos organiza los archivos de forma contigua utilizando registros lógicos de 100 bytes y registros físicos de 500 bytes.

La memoria secundaria (el disco duro) está formada por una sola superficie con seis pistas y cuatro sectores (registros) por pista, por tanto, se tienen veinticuatro registros numerados [0-23].

Pistas	NTRP	TRF
6	4	500

Ejemplo

Distribución de los registros dentro del disco.



Ejemplo

- 1) Se atiende la solicitud del usuario (`fopen("prueba", "r")`), se lee el archivo 'prueba' en el número de registro lógico 9 en el registro físico en la localidad 14,000.

Ejemplo

Pistas	NTRP	TRF	NRL	Localidad
6	4	500	9	14,000

2) Sistema de archivos simbólicos. Utiliza el directorio de archivos simbólicos para transformar el nombre prueba en un identificador interno, de la siguiente manera:

Archivo simbólico	Identificador interno
xyz	3
prueba	2
maestro	6

Ejemplo

Pistas	NTRP	TRF	NRL	Localidad	ID
6	4	500	9	14,000	2

3) Sistema de archivos básico. Utiliza el identificador interno (en este caso 2) para dar acceso al directorio de archivos básicos y copiar la información contenida en el archivo en la memoria principal.

Identificador interno	Tamaño del registro lógico	Registros lógicos que forman el archivo	Dirección del primer registro lógico
2	100	10	9
3	500	1	20
4	50	10	4

Ejemplo

4) Sistema de archivos lógicos. Toma como entrada el número de registro lógico (NRL) y el tamaño del mismo (TRL) y emite como salida la dirección lógica de la cadena de bytes (DLCB) y el tamaño de la misma (TCB).

Para esto utilizará la información del directorio de archivos básicos y las siguientes fórmulas:

$$\begin{aligned} \text{DLCB} &= (\text{NRL} - 1) * \text{TRL} \\ \text{TCB} &= \text{TRL} \end{aligned}$$

Ejemplo

Pistas	NTRP	TRF	NRL	Localidad	ID
6	4	500	9	14,000	2
TRL	NRLA	DPRL			
100	10	9			

Sustituyendo en el ejemplo se tiene:

$$\text{DLCB} = (\text{NRL} - 1) * \text{TRL}$$

$$\text{DLCB} = (9 - 1) * 100 = 800$$

$$\text{TCB} = \text{TRL}$$

$$\text{TCB} = 100$$

Ejemplo

5) Sistema de archivos físicos. Toma como entrada la dirección lógica de la cadena de bytes (DLCB), el tamaño del registro físico (TRF) y la dirección del primer registro físico (DPRF) para dar como salida el número de registro físico (NRF) y el desplazamiento(D).

La transformación se lleva a cabo utilizando las siguientes fórmulas:

$$\begin{aligned} \text{NRF} &= (\text{DLCB} \text{ div } \text{TRF}) + \text{DPRF} \\ \text{D} &= (\text{DLCB} \text{ mod } \text{TRF}) \end{aligned}$$

Ejemplo

Pistas	NTRP	TRF	NRL	Localidad	ID
6	4	500	9	14,000	2
TRL	NRLA	DPRL	DLCB	TCB	
100	10	9	800	100	

Sustituyendo en las fórmulas anteriores para el archivo prueba se tiene:

$$NRF = (DLCB \text{ div } TRF) + DPRF$$

$$NRF = (800 \text{ div } 500) + 9 = 10$$

$$D = (DLCB \text{ mod } TRF)$$

$$D = (800 \text{ mod } 500) = 300$$

Ejemplo

6) Sistema de manejo del dispositivo. Este módulo toma como entrada el NRF y lo transforma en una dirección compuesta por el número de pista (NP) y el número de registro dentro de la pista (NRDP).

El sistema de manejo del dispositivo utiliza el número total de pistas (NTP) para obtener el número total de registros en una pista (NTRP) de acuerdo a las siguientes fórmulas:

$$\begin{aligned} \text{NP} &= (\text{NRF} \text{ div } \text{NTRP}) \\ \text{NRDP} &= (\text{NRF} \text{ mod } \text{NTRP}) \end{aligned}$$

Ejemplo

Pistas	NTRP	TRF	NRL	Localidad	ID	
6	4	500	9	14,000	2	
TRL	NRLA	DPRL	DLCB	TCB	NRF	D
100	10	9	800	100	10	300

Sustituyendo en las fórmulas se tiene:

$$NP = (NRF \text{ div } NTRP)$$

$$NP = (10 \text{ div } 4) = 2$$

$$NRDP = (NRF \text{ mod } NTRP)$$

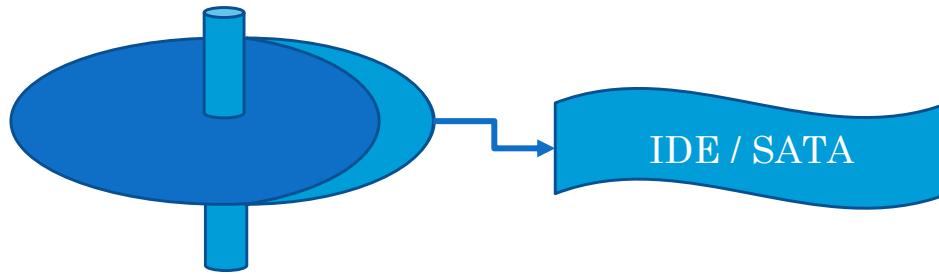
$$NRDP = (10 \text{ mod } 4) = 2$$

Ejemplo

Pistas	NTRP	TRF	NRL	Localidad	ID			
6	4	500	9	14,000	2			
TRL	NRLA	DPRL	DLCB	TCB	NRF	D	NP	NRDP
100	10	9	800	100	10	300	2	2

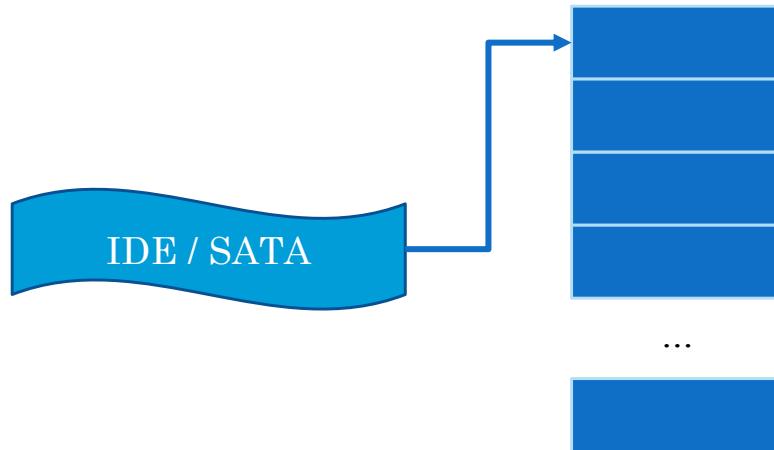
Ejemplo

Con esta información se puede ejecutar una instrucción para que la unidad de disco lea el segundo registro físico de la pista 2 y lo transfiera al buffer en la memoria principal.



Ejemplo

Una vez que se tenga los registros físicos en la memoria principal, el sistema operativo utilizará el desplazamiento D calculado para extraer la información correspondiente al registro lógico del usuario (9) y copiarla a partir de la dirección 14,000.



Archivos

Python

Lectura y escritura de archivos en Python

Para leer un archivo se requiere crear una conexión entre el programa en la memoria RAM y el archivo en la memoria secundaria. Para establecer este canal de comunicación se utiliza la función open en Python.

Open

La función open se puede utilizar con los siguientes parámetros:

```
open(file, mode='r', buffering=-1, encoding=None, errors=None,  
      newline=None, closefd=True, opener=None)
```

file es la ruta donde se encuentra o donde se va a escribir el archivo. Si no se agrega el nodo raíz, se toma como ruta relativa, si se adjunta, se toma como ruta absoluta.

```
open(file, mode='r', buffering=-1, encoding=None, errors=None,  
      newline=None, closefd=True, opener=None)
```

mode es un parámetro opcional que especifica la forma en la que se va a abrir el archivo. Los modos disponibles son:

Carácter	Significado
'r'	Abre para lectura (default).
'w'	Abre para escritura.
'x'	Abre exclusivamente para creación, falla si el archivo ya existe.
'a'	Abre para escritura, agrega al final del archivo si éste ya existe.
'b'	Modo binario.
't'	Modo texto (default).
'+'	Abre un archivo del disco para actualizar (tanto en modo lectura como escritura).
'U'	Modo nueva línea universal (obsoleto).

```
open(file, mode='r', buffering=-1, encoding=None, errors=None,  
      newline=None, closefd=True, opener=None)
```

buffering es un entero opcional que permite establecer la política del buffer. 0 apaga el buffer (solo para archivos en modo binarios), 1 selecciona un buffer de línea (solo para archivos en modo texto) y un entero mayor a 1 indica el número de bytes que puede leer el buffer.

```
open(file, mode='r', buffering=-1, encoding=None, errors=None,  
      newline=None, closefd=True, opener=None)
```

encoding es un parámetro opcional que se refiere al tipo de codificación que se utilizará para leer y escribir el archivo. Este valor se utiliza cuando se abre el archivo en modo texto. La codificación por defecto se obtiene del sistema utilizando la función `locale.getpreferredencoding()`.

```
open(file, mode='r', buffering=-1, encoding=None, errors=None,  
      newline=None, closefd=True, opener=None)
```

errors es una cadena opcional que especifica cómo codificar y decodificar los errores (no se puede usar en modo binario).

```
open(file, mode='r', buffering=-1, encoding=None, errors=None,  
      newline=None, closefd=True, opener=None)
```

newline controla como trabaja el modo nueva línea universal (solo para modo de texto). Su valor puede ser None, ‘‘, ‘\n’, ‘\r’, ‘\r\n’.

```
open(file, mode='r', buffering=-1, encoding=None, errors=None,  
      newline=None, closefd=True, opener=None)
```

opener un descriptor de archivo se obtiene a través de un opener, el cual recibe los parámetros archivo y banderas para crear el descriptor.

Ejemplo

```
# -*- coding: utf-8 -*-
import locale

try:
    file = open("open.py", "r", encoding='utf-8')
    print(locale.getpreferredencoding())
    print(locale.setlocale(locale.LC_ALL, ""))
except:
    print('No fue posible abrir el archivo.')
finally:
    if file:
        file.close()
```

with

La sentencia `with` se utiliza para envolver la ejecución de un bloque de instrucciones con métodos definidos por el administrador de contexto, generando un entorno de ejecución seguro. Si un archivo se abre con esta instrucción, el administrador de contexto se encargará de cerrar el flujo del archivo al finalizar el bloque en ejecución.

Ejemplo

```
# -*- coding: utf-8 -*-
import locale

with open("open.py", "r", encoding='utf-8') as file:
    print(locale.getpreferredencoding())
    print(locale.setlocale(locale.LC_ALL, ''))
```

Properties

Una vez abierto un archivo se pueden conocer algunos datos del mismo, como lo son el nombre, el modo con el que fue abierto e, incluso, la codificación dentro del sistema.

Ejemplo

```
# -*- coding: utf-8 -*-

file = open("attributes.py", 'r')
name    = file.name
mode    = file.mode
encoding = file.encoding
file.close()

print("Name",name)
print("Mode",mode)
print("Encoding",encoding)
print("Closed file?", file.closed)
```

read

La función `read(size)` lee cierta cantidad de información y regresa una cadena (en modo texto) o un objeto de bytes (en modo binario). El parámetro es opcional, cuando se omite o es negativo, se regresa el contenido completo del archivo, cuando se `size` es positivo, se lee la `size` bytes a la vez.

Ejemplo

```
# -*- coding: utf-8 -*-
import locale

print("read()")
with open("read.py", "r", encoding='utf-8') as file:
    print(file.read())

print("\nread(-111)")
with open("read.py", "r", encoding='utf-8') as file:
    print(file.read(-111))

print("\nread(10)")
with open("read.py", "r", encoding='utf-8') as file:
    print(file.read(10))
```

Ejemplo

```
print("\nread(10) binary")
with open("read.py", "rb") as file:
    print(file.read(10))

with open("read.py", "r") as file:
    while True:
        line = file.readline()
        if not line:
            break
        print (line)
```

readline

La función readline() lee una línea del archivo. Agrega un salto de línea a todas las líneas del archivo, excepto a la última. Si readline() regresa un línea vacía, quiere decir que se llegó al final del archivo, una línea en blanco en el archivo se lee como '\n'.

Ejemplo

```
# -*- coding: utf-8 -*-

print("readline()")
with open("readline.py", "r", encoding='utf-8') as file:
    print(file.readline())

with open("readline.py", "r", encoding='utf-8') as file:
    for line in file:
        print(line)

with open("readline.py", "r", encoding='utf-8') as file:
    print(list(file))

with open("readline.py", "r", encoding='utf-8') as file:
    print(file.readlines())
```

write

La función `write(string)` escribe el contenido de `string` en el archivo, devuelve el número de caracteres escritos.

Tipos de datos diferente a `string` deben convertirse utilizando la función `str(value)`, ya sea en modo texto o binario.

La función `writelines(list)` guarda todos los datos de la lista en el archivo, los elementos de la lista deben ser de tipo cadena.

Ejemplo

```
# -*- coding: utf-8 -*-
numbers = ['hello', 'world']
print("write()")

with open("file.txt", "w") as file:
    print(file.write("Hello world!"))
    print(file.writelines(numbers))

with open("file.txt", "r") as file:
    print(file.readlines())
```

tell

La función tell() regresa la posición actual del objeto file en el archivo (apuntador), representado por el número de bytes o caracteres desde el inicio del archivo.

Ejemplo

```
# -*- coding: utf-8 -*-
print("tell()")
with open("tell.py", "r") as file:
    print(file.tell())
    print(file.readline())
    print(file.tell())
```

seek

La función seek(offset, from_what) permite cambiar la posición del objeto file en el archivo. Al punto de referencia se le agrega el offset enviado, el punto de referencia se obtiene del argumento from_what (opcional). Los valores de from_what son 0 para el inicio del archivo (por defecto), 1 para la posición actual del apuntador y 2 para el final del archivo

Ejemplo

```
# -*- coding: utf-8 -*-
print("seek()")
with open("seek.py", "r") as file:
    print(file.tell())
    print(file.readline())
    print(file.tell())
    print(file.seek(30))
    print(file.readline())
    print(file.seek(3, 0))
    print(file.readline())
# print(file.seek(-2, 2))
# print(file.readline())
```

pickle

La biblioteca pickle también permite manipular archivos. La función dump(data, file) permite escribir la lista de datos data en el archivo file. La función load(file) lee los datos del archivo en forma de cadena.

Ejemplo

```
# -*- coding: utf-8 -*-
import pickle

print("pickle")
memory = ['Hello','world','!!']
file = open('list.txt', 'wb')
pickle.dump(memory, file)
file.close()

file = open('list.txt', 'rb')
file_memory = pickle.load(file)
file.close()

print(file_memory)
```

os

La biblioteca os provee funciones para operar directamente con el sistema operativo nativo. Esta biblioteca posee diversos métodos muy interesantes, los cuales se pueden revisar en:

<https://docs.python.org/3.6/library/os.html>

os.mkdir

La función `mkdir(path, mode=0o777, *, dir_fd=None)` crea la carpeta con el nombre path y el nivel de acceso mode. Si la carpeta ya existe se regresa un error del tipo `FileExistsError`.

os.chdir

La función chdir(path) cambia el directorio de trabajo actual por la ruta enviada en path.

os.makedirs

La función makedirs(name, mode=0o777, exist_ok=False) crea las carpetas intermedias necesarias para contener a la carpeta hoja. Si exist_ok es False (por defecto), se arroja un error del tipo OSError si la carpeta ya existe.

Ejemplo

```
# -*- coding: utf-8 -*-
import os

def make_dir(path):
    try:
        os.mkdir(path)
    except:
        print("Error al crear el archivo", path)
    os.chdir(path)

print("os.mkdir")
make_dir("test1")
```

os.walk

La función `walk(top, topdown=True, onerror=None, followlinks=False)` genera los nombres de los archivos de un árbol de carpetas recorriendo el árbol ya sea de arriba hacia abajo (top-down) o de abajo hacia arriba (bottom-up). Para cada carpeta se regresa una tupla de tres valores: (`dirpath`, `dirnames`, `filenames`).

Ejemplo

```
# -*- coding: utf-8 -*-
import os

def dir_walk(root_dir):
    for dir_path, dir_names, file_names in os.walk(root_dir):
        print("Carpeta,", dir_path)
        print("Subcarpetas,", dir_names)
        print("Archivos,", file_names)

print("os.walk")
dir_walk(".")
```

Cadena

Una cadena es una secuencia de caracteres. Para declarar una cadena se puede utilizar comillas dobles o comillas simples:

‘Lenguaje Python’
“Lenguaje Python”

Subcadena

Python permite manejar las cadenas como arreglos, es decir, se pueden utilizar índices para obtener caracteres específicos de la cadena.

cadena[<expresión>]

También se puede obtener una sección de la cadena especificando el rango deseado.

```
cadena[<inicio_expresión>:<fin_expresión>]
```

La subcadena que se obtiene va del inicio_expresión hasta un carácter antes de fin_expresión.

Si no se especifica la expresión final, se obtiene la cadena desde inicio_expresión y hasta el final de la cadena.

Búsquedas en cadenas

La función `find()`, sirve para buscar un texto dentro de una cadena, regresa la posición donde inicia la coincidencia del texto en la cadena.

Si no hay coincidencia del texto en la cadena la función regresa el valor -1.

También se puede especificar un índice a partir de donde se desea iniciar la búsqueda dentro de la cadena.

`cadena.find("patrón", índice)`

Ejemplo

```
# -*- coding: utf-8 -*-

# ----- Búsquedas en cadenas -----

# Cadena original
cadena = "Amor y deseo son dos cosas diferentes; que no todo lo
que se ama se desea, ni todo lo que se desea se ama. (Don
Quijote)"

# Busca la cadena "ama"
print (cadena.find("ama"))

#Imprimie la subcadena
print (cadena[cadena.find("ama"):])
```

Ejemplo

```
# Busca la siguiente coincidencia de "ama"  
print (cadena[cadena.find("ama") + 1:]).find("ama"))
```

```
# Imprime la cadena a partir de la segunda coincidencia  
print (cadena[61 + 1 + 40:])
```

```
# Busca "corazon" en la cadena  
print (cadena.find("corazon"))
```

```
# Busca a partir de un indice  
print (cadena.find("todo", 62))
```

```
# Imprime a partir de un índice y hasta el final  
print (cadena[78:])
```

urllib

La biblioteca `urllib` permite manejar diversas peticiones html. La función `request` permite hacer la solicitud de una url para obtener el código HTML contenido.

Ejemplo

```
# -*- coding: utf-8 -*-

import urllib.request

def getHTML(strURL):
    return str(urllib.request.urlopen(strURL).read())

def main():
    url = 'http://odin.fi-b.unam.mx/python/'
    html = getHTML(url)
    print(html)

main()
```

Ejemplo

```
# -*- coding: utf-8 -*-

import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

impr = ["Aprobados", "No aprobados", "deserción", "NP"]
vol = [2, 10, 20, 10]
expl =(0.05, 0.05, 0, 0)
fig1, ax1 = plt.subplots()
ax1.pie(vol, explode=expl, labels=impr, autopct='%.1f%%',
shadow=True,startangle=90)
plt.show()
```

Archivos.

Práctica 10

10. Archivos.

- A partir de un xml con los siguientes datos:

```
<record>
  <user>Matsoft</user>
  <password>Avery</password>
  <email>achapleo0@npr.org</email>
</record>
```

- Crear una función que permita obtener todos los datos almacenados en éste, generando una lista de diccionarios:

```
[{user = Matsoft, password = Avery,
  email = achapleo0@npr.org}]
```

10. Archivos.

- Una vez terminada la búsqueda, generar las siguientes estadísticas y guardarlas en disco:
 - Número y lista de passwords iguales.
 - Número y password con mayor incidencia.
 - Número de empresas (dependiendo del correo).
 - Número y lista de correos iguales.
- Graficar el tiempo de ejecución de las funciones anteriores.
- Generar una gráfica de pie con los números obtenidos de las funciones.

<https://mockaroo.com>



5 Archivos

Objetivo: Interpretar las organizaciones básicas de los archivos, las operaciones que se pueden realizar sobre ellos y su representación mediante diferentes medios de almacenamiento secundario.

5.1 Generalidades.

5.2 Definición y operaciones.

5.3 Organización de archivos.

5.3.1 Organización lógica.

5.3.2 Organización física.

5.4 Acceso a archivos.

5.4.1 Acceso lógico.

5.4.2 Acceso físico.

5.5 Sistema de archivos.