

Tema 4: Administración de Archivos y Disco

Universidad de Las Américas
Facultad de Ingeniería y Negocios
Prof. **Iván Jiménez Utiel**
ACI 343

June 24, 2014

Contents

1	Almacenamiento masivo y planificación de disco	1
1.1	Almacenamiento masivo	2
1.2	Disk Scheduling	3
1.2.1	FCFS	4
1.2.2	SSTF	4
1.2.3	SCAN	5
1.2.4	C-SCAN	5
2	Interfaz del Sistema de Archivos	6
2.1	Archivos	6
2.2	Estructura de Directorios	8
2.2.1	Árbol de los Directorios	9
2.3	Protección y Seguridad	10
3	Implementación del Sistema de Archivos	12
3.1	Sistemas de Archivos	12
3.2	Unix Volume Management	13
3.3	Virtual File System (Linux)	14
4	Taller de Laboratorio	15
5	Bibliografía	15

1 Almacenamiento masivo y planificación de disco

En este tema vamos a ver ciertos aspectos sobre la gestión de archivos atendiendo a aspectos de administración y gestión mediante el Sistema Operativo.

Definición 1.1 (Data Rate). Tasa de Transferencia de datos de un dispositivo I/O.

Podemos ver en la siguiente imagen cómo el **Data Rate** es mucho más significativo en dispositivos I/O como la red GigaEthernet o en Dispositivo de Gráficos.

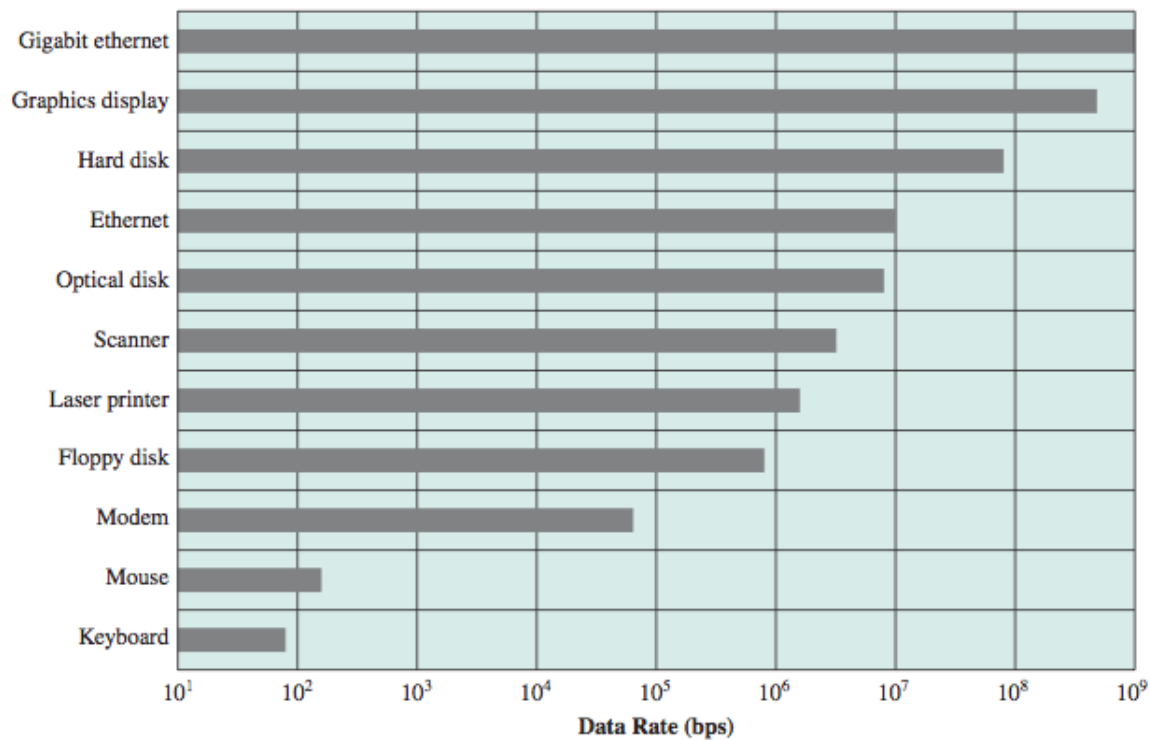


Figure 11.1 Typical I/O Device Data Rates

1.1 Almacenamiento masivo

Los discos magnéticos tienen la siguiente estructura básica:

- Uno o más platos (platters) en forma de discos cubiertos con material magnético. El Hard Disk (HD) tiene los platos de metal (los Floppys antiguos eran de plástico).
- Cada plato tiene dos superficies de trabajo → las unidades de discos antiguos sólo utilizaban una sola cara (side), aunque eran más susceptibles a daños.
- Cada superficie de trabajo es dividida en anillos llamados **tracks**. La colección de estos tracks que están a la misma distancia del eje del plato son llamados cilindros (**cylinder**).
- Cada track se divide en **sectores** (tradicionalmente de 512 bytes cada uno, los modernos tienen más). Los sectores tienen un header que incluyen información de **checksum**.
- Los datos en un Hard Disk es puede ser leída por las cabezas. En la figura poemas ver que hay una cabeza por superficie con un brazo (arm) y controlado por un **arm assembly** el cual mueve todas las cabezas de forma simultánea de un cilindro a otro → Que tenga cabezas independientes aumenta la velocidad pero también dificultades técnicas.

Capacidad de disco = heads * nº tracks por superficie * nº sectores por track * número de bytes por sector

- La velocidad de transferencia del disco está limitada por el tiempo de búsqueda (seek times) y la latencia de rotación (rotación latency). Cuando hay múltiples peticiones que son procesadas existe un retardo en la espera.
- El ancho de banda se mide por la cantidad de datos transferidos dividido por el total de tiempo de la

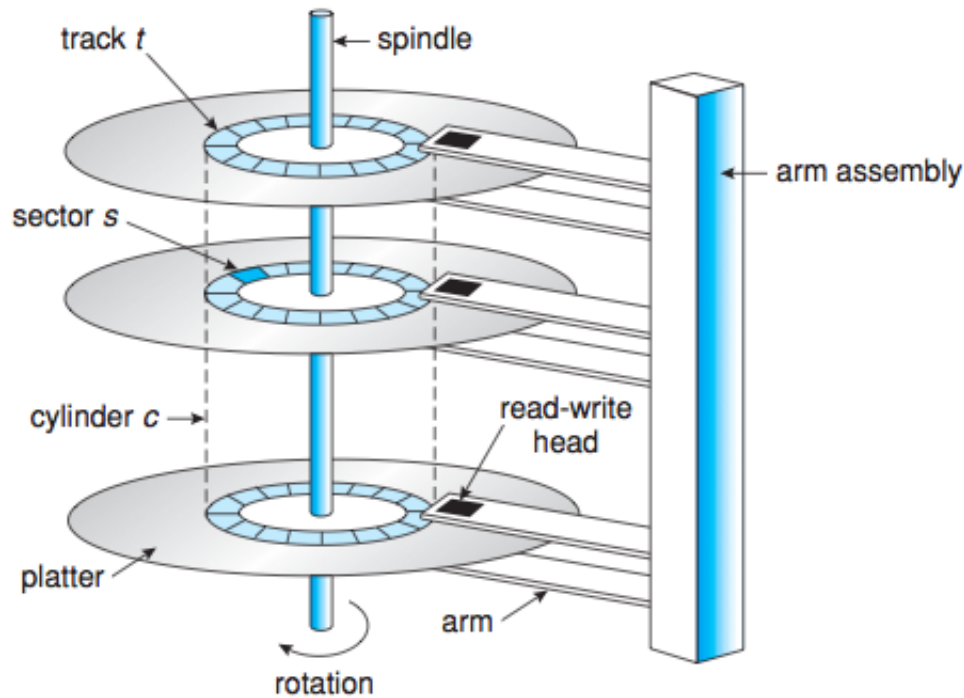


Figure 10.1 Moving-head disk mechanism.

1.2 Disk Scheduling

En una operación de disco a alta velocidad 7200 rpm (120 revoluciones por segundo). La tasa en la que se transfieren los datos del disco a la computadora tiene los siguientes pasos:

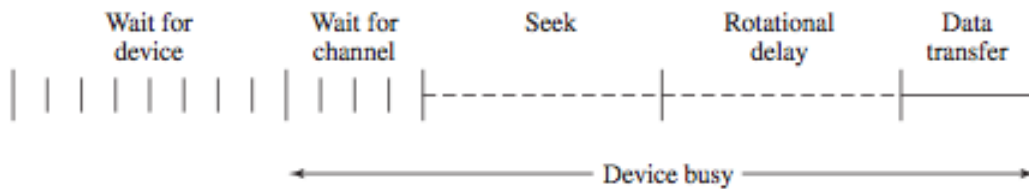


Figure 11.6 Timing of a Disk I/O Transfer

- La velocidad de transferencia está limitada por el **Seek time** y el **Rotational delay**.
- El **Bandwith** o ancho de bando es la cantidad de datos transferidos por el total de tiempo desde la primera hasta la última petición completada.
- Tanto el bandwith como el tiempo de acceso pueden ser mejoradas mediante el procesamiento de las peticiones escogiendo un buen orden.
- Las peticiones de discos incluyen direcciones de disco, de memoria, número de sectores a transferir y si es una petición R o WR.

En la siguiente sección supongamos que tenemos un disco y una cola de peticiones. La cabeza del disco está inicialmente en la 53. La lista de peticiones es la siguiente: 98, 183, 37, 122, 14, 124, 65, 67.

1.2.1 FCFS

- Es el más simple.
- Las peticiones son procesadas en orden de llegada secuencial de la lista .
- Observa como de la petición 122 va a la 14 y luego a la 124!!!

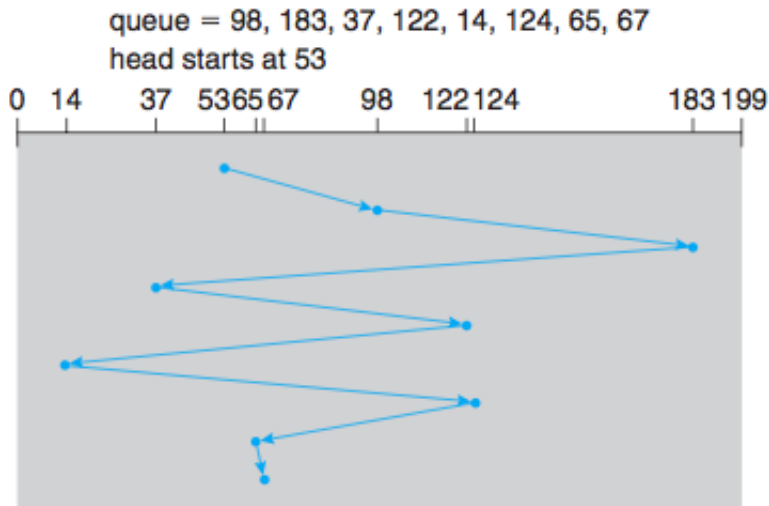


Figure 10.4 FCFS disk scheduling.

Podemos ver con la consola de Python cómo restando los valores de cada petición en la lista conforme son procesadas, se puede obtener el número total que ha tenido que recorrer el cabezal:

```
$ python
>>> abs(53-98)+abs(183-98)+abs(37-183)+abs(37-122)+abs(14-122)+abs(124-14)+abs(124-65)+abs(65-67)
640
>>>
```

1.2.2 SSTF

- Shortest Seek Time First es más eficiente, pero puede llevar a inanición si la cola de peticiones llega para el mismo área de disco.
- SSTF reduce el movimiento a 236 peticiones, mucho menos que las 640 para el mismo número de peticiones en FCFS. Nótese que la distancia podría ser reducida hasta 208 comenzando en la 37 y luego en la 14 para después procesar el resto de las peticiones.

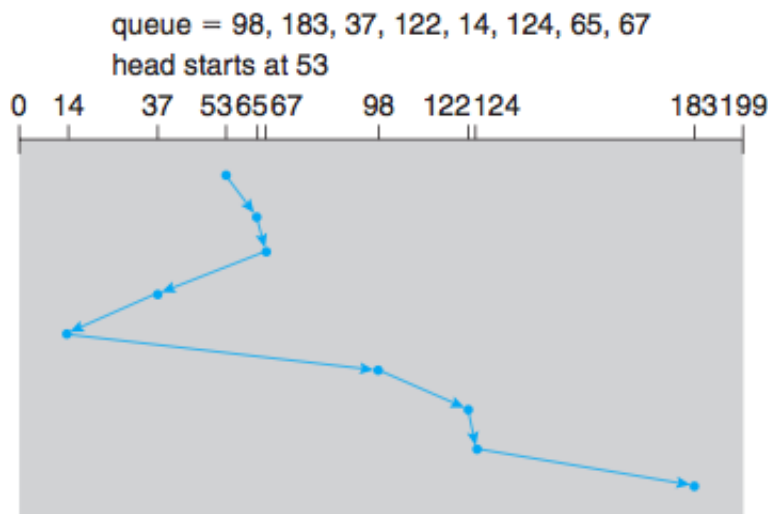


Figure 10.5 SSTF disk scheduling.

1.2.3 SCAN

- El brazo comienza a desplazarse hasta el principio del disco (asumimos que se mueve hacia el 0) sirviendo las peticiones que encuentra y, cuando llega al extremo, se va hacia el otro extremo sirviendo de igual manera las peticiones. Este algoritmo también se llama **ascensor** o **elevador**.
- Vemos que, hay peticiones que en un extremo pueden tener un tiempo de espera alto: 122, 124, 199.

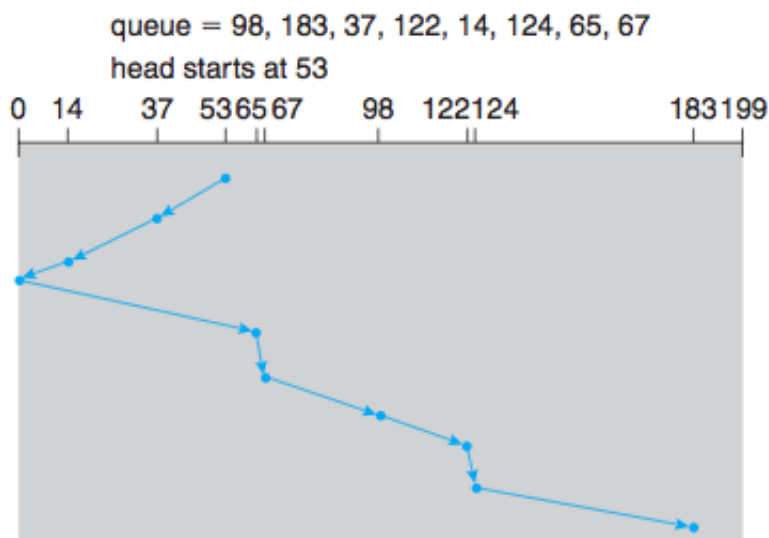


Figure 10.6 SCAN disk scheduling.

1.2.4 C-SCAN

- Circular SCAN: es una variante de SCAN diseñada para mejorar el tiempo de espera.
- La cabeza se desplaza hacia un extremo del disco sirviendo todas las peticiones que encuentre.
- Al llegar al final, vuelve al otro extremo **SIN** atender las peticiones que se encuentre.

- En ese momento vuelve a desplazarse la cabeza en la misma dirección, atendiendo a las peticiones que se encuentren.
- En el ejemplo vemos como se atienden las peticiones hasta la 199 y luego vuelve la cabeza sin procesar nada hasta la 0. Ahí comienza otra vez a atender las peticiones.

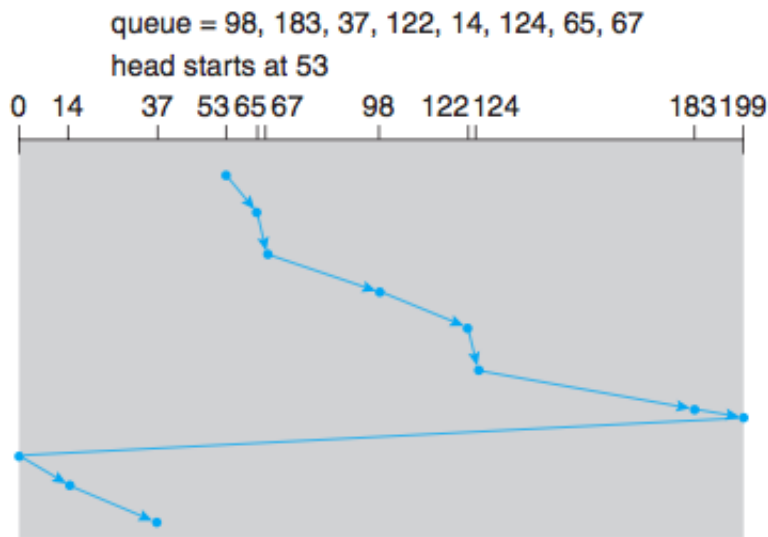


Figure 10.7 C-SCAN disk scheduling.

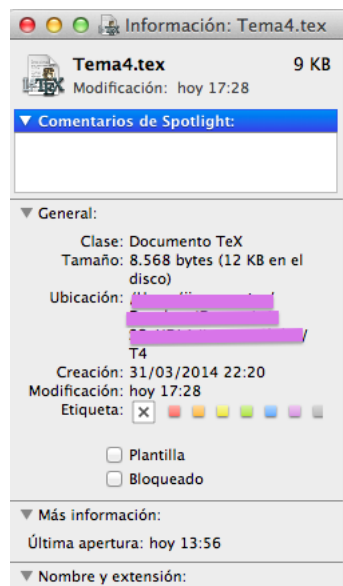
Selección del mejor algoritmo de scheduling:

- Con baja carga, todos los algoritmos funcionan casi de igual manera.
- Por cargas un poco más altas, SSTF funciona mejor que FCFS, aunque tiende a tener problemas de inanición (starvation).
- Para sistemas sobrecargados, SCAN y C-SCAN son buenas elecciones, aunque existen mejores (más costosas) implementaciones.

2) Interfaz del Sistema de Archivos

2.1 Archivos

- Diferentes SO mantienen una serie de atributos de archivos, incluyendo:
 - Name: algunos sistemas incluyen extensiones (.exe, .txt, .png)
 - Identifier: (número de i-nodo).
 - Type: texto, ejecutable, binario, etc.
 - Location
 - Size
 - Protection
 - Time&Date
 - User ID



- Operaciones con archivos: create, write, read, deleting, truncate.
- Muchos SOs requieren que el fichero esté abierto para que pueda ser utilizado y cerrado después de su uso. Normalmente es el programador el que debe realizar estas operaciones.
- La información de los archivos abiertos actualmente es guardada en una tabla de archivos abiertos que contienen:
 - Puntero de Fichero: guarda la posición actual.
 - Contador de Fichero abierto: cuántas veces ha sido abierto y si lo está aún.
 - Ubicación en disco:
 - Derechos de Acceso.

Podemos ver en este par de ejemplos ciertas funciones sobre archivos:

```
##En Python
$ python
>>> import os
>>> path = 'server.py'
>>> f = open(path, 'rb')
>>> os.path.getsize(path)
110

>>> os.stat(path)
(33204, 37366188L, 143L, 1, 501, 501, 285L, 1402324550, 1400086770, 1400086770)
>>>

>>> os.getcwd()
'/home/vegas'
>>>

>>> os.mkdir("/home/vegas/dir1")
>>> os.listdir("/home/vegas/dir1")
[]
>>> os.mkdir("/home/vegas/dir1/dir2")
>>> os.listdir("/home/vegas/dir1")
['dir2']
>>>

## En Shell
$ stat fork_ping.py
  File: ■fork_ping.py■
  Size: 285          Blocks: 8          IO Block: 4096    fichero regular
Device: 8fh/143d    Inode: 37366188    Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 501/ijimenez)   Gid: ( 501/ijimenez)
Access: 2014-06-09 18:35:50.610411656 +0400
Modify: 2014-05-14 20:59:30.442261098 +0400
Change: 2014-05-14 20:59:30.443261098 +0400
```

- Windows (y otros sistemas utilizan extensiones de fichero especiales para indicar qué tipo de fichero utilizan).

file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, perl, asm	source code in various languages
batch	bat, sh	commands to the command interpreter
markup	xml, html, tex	textual data, documents
word processor	xml, rtf, docx	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	gif, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	rar, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, mp3, mp4, avi	binary file containing audio or A/V information

- MACOS almacena un atributo por cada fichero de acuerdo al programa que lo creó.
- UNIX/LINUX/POSIX almacena número mágicos al comienzo de ciertos archivos. Probemos el comando **file**:

```
$ file /dev/ram
/dev/ram: symbolic link to 'ram1'

$ file /dev/pts/0
/dev/pts/0: character special (136/0)

$ file /bin/grep
/bin/grep: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), for GNU/Linux 2.6.9,
dynamically linked (uses shared libs), stripped
```

2.2 Estructura de Directorios

- Un disco puede ser utilizado en su totalidad por un sistema de archivos.
- Alternativamente un disco físico puede ser dividido en varias particiones, porciones, o mini discos. Cada uno de los cuales puede ser un disco virtual y puede tener su propio sistema de archivos (o también utilizado para **raw store** o **swap space**).
- También puede ser que discos físicos múltiples puedan ser combinados en un **volumen** como si fueran un disco más grande, con su propio sistema de archivos abarcando todos los discos físicos.

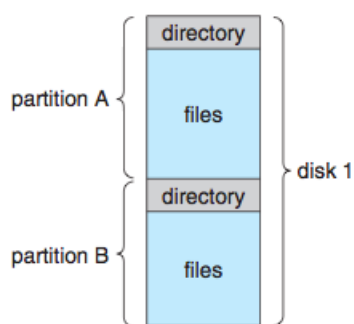


Figure 11.7 A typical file-system organization.

/	ufs
/devices	devfs
/dev	dev
/system/contract	ctfs
/proc	proc
/etc/mnttab	mntfs
/etc/svc/volatile	tmpfs
/system/object	objfs
/lib/libc.so.1	lofs
/dev/fd	fd
/var	ufs
/tmp	tmpfs
/var/run	tmpfs
/opt	ufs
/zpbge	zfs
/zpbge/backup	zfs
/export/home	zfs
/var/mail	zfs
/var/spool/mqueue	zfs
/zpbg	zfs
/zpbg/zones	zfs

Figure 11.8 Solaris file systems.

2.2.1 Árbol de los Directorios

- El uso de una estructura de directorios en árbol minimiza la dificultad en asignar nombres únicos.
- Cada archivo del sistema puede ser accedido en el path (o camino) desde el root (también directorio máster o raíz) bajando por las distintas ramas hasta llegar al archivo.
- El camino que va desde el raíz hasta el archivo, se llama **pathname**.
- Ejemplo de pathname para el archivo ABC: **/User_B/Word/Unit_A/ABC**
- Ejemplo de pathname para el archivo ABC: **/User_B/Draw/ABC**
- Nótese que están en distintos pathnames aunque sean iguales los archivos.

Es interesante la utilización de estos dos comandos en Linux sobre uso y estadísticas de disco

Comando df espacio utilizado en los discos

```
$ df -h
Filesystem      Size  Used Avail Capacity  Mounted on
/dev/disk0s2    74Gi  62Gi   11Gi    85%      /
devfs           111Ki  111Ki   0Bi   100%     /dev
map -hosts      0Bi    0Bi   0Bi   100%     /net
map auto_home   0Bi    0Bi   0Bi   100%     /home
```

Estadísticas de uso del disco

```
$ du -ah
8,0K    ./ .DS_Store
16K     ./ ipcop_network.png
4,0K    ./ p21.aux
```

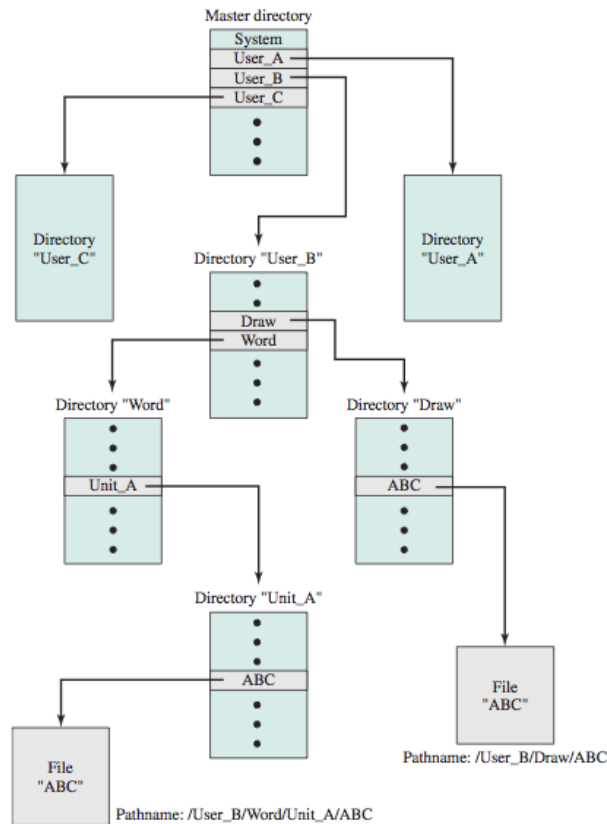


Figure 12.7 Example of Tree-Structured Directory

```

24K    ./p21.log
0B     ./p21.out
132K   ./p21.pdf
156K   ./p21.synctex.gz
24K    ./p21.tex
12K    ./udla.png
376K   .

```

2.3 Protección y Seguridad

- Los archivos deben mantenerse a salvo por seguridad (daños accidentales) y protección (contra accesos maliciosos: virus, hacks, etc.).
- Aparte de realizar copias de backup, un simple esquema de protección elimina todo acceso a fichero. Sin embargo, esto hace al fichero inutilizable, con lo que un acceso **controlado** y gestionado hace que sea una mejor opción.

Tipo de Acceso:

- **Operaciones de bajo nivel:**
 - Read: ver los contenidos del archivo.
 - Write: cambiar los contenidos del archivo.
 - Execute: cargar el fichero en la CPU y seguir las instrucciones que contiene.
 - Append: Añadir al final del archivo.
 - Delete: Eliminar el archivo del sistema.
 - List: Ver el nombre y otros atributos.

- Operaciones de alto nivel: copy, tar, zip, etc.

#Ejemplo de empaquetado TAR mediante Python

```
>>> f = open("largefile.txt","w")
>>> statement = "Esto es una gran línea"
>>> x=0
>>> for x in xrange(2000):
...     x += 1
...     f.write("%s\n" % statement)
...
>>>
>>> import tarfile
>>> tar = tarfile.open("largefile.tar","w")
>>> tar.add("largefile.txt")
>>> tar.close()
>>>
```

Control de Acceso:

- Un enfoque sería tener complicadas ACLs (Access Control Lists), las cuales especifican exactamente qué acceso es permitido o denegado para cada usuario o grupos específicos. → Realmente es bastante complicada su administración aunque muy precisa.
- Otro enfoque: UNIX/LINUX utiliza un conjunto de 9 bits de control en tres grupos de tres. Estos corresponden a los permisos R, W y X para cada u(ser), g(rupo), o(thers). La combinación de RWX controla los privilegios de los archivos y directorios:

Bit	Files	Directories
R	Read (ver el contenido del archivo)	Leer el contenido de los directorios. Requerido para listar el contenido de un directorio.
W	Write (cambiar el contenido del archivo)	Cambia el contenido del directorio. Requerido para crear o eliminar archivos.
X	eXecution (ejecuta el fichero como un programa)	Información detallada de la información. Requerida para listar el contenido de forma extendida o acceso a determinados archivos*.

(* Nótese que si el usuario tiene permisos X pero no permisos de R en un directorio, puede acceder a un determinado archivo sí y sólo sí, conocen el nombre del archivo al que quieren acceder).

Tal y como podemos apreciar en la siguiente salida (ls -l) de listado de directorio tipo UNIX/Linux el primer campo (o columna) describe la protección del fichero o directorio. El primer carácter puede ser **d**(directorio) ó **-**(archivo regular), a continuación hay una combinación de 9 caracteres agrupados en grupos de tres (rwx) que corresponden a los privilegios de **usuario, grupo y otros** respectivamente.

Después se muestran el número de enlaces al archivo, el propietario del mismo, después el nombre del grupo, el tamaño del archivo en bytes, la fecha de la última modificación y finalmente el nombre del archivo.

```
drwxrwxr-x  2 ijimenez ijimenez  4096 jun 10 03:22 bash_cookbook
-rw-r--r--  1 root      root      738702 jun 20 20:37 default.tar.gz
drwxrwxr-x 43 ijimenez ijimenez  4096 may  8 02:19 dev8082
drwxrwxr-x  2 ijimenez ijimenez  4096 jun 10 04:18 diveinpython
-rw-r--r--  1 root      root      10474 jun 15 21:00 ez_setup.py.1
```

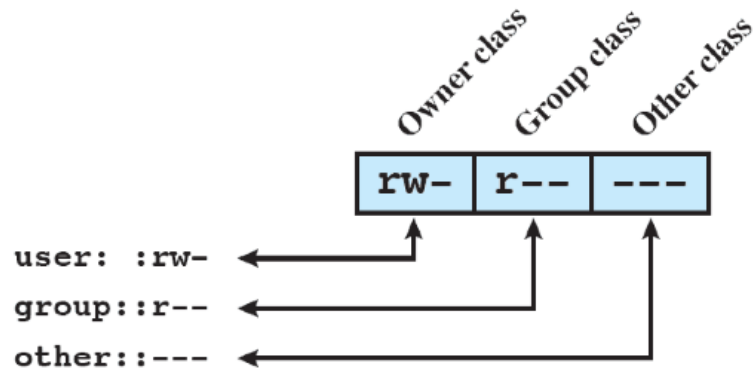
Podemos ver también un esquema de esos privilegios de forma más gráfica en la figura de arriba.

Con el comando **chmod** (**man chmod** para más información) se pueden administrar los privilegios de los archivos/directorios:

```
-rwxr-xr-x  1 wada  users  4096 abr 13 19:30 archivo

$ chmod g-x,o-x archivo # quitaremos el privilegio de ejecución a los del grupo y a otros
# quedando:
-rwxr--r--  1 wada  users  4096 abr 13 19:30 archivo

$ chmod u-x archivo # Daremos privilegios de ejecución al usuario para \textbf{archivo}
# quedando:
-r-xr--r--  1 wada  users  4096 abr 13 19:30 archivo
```



También se puede hacer utilizando numeración binari-octal (fijaos que en la primera columna se utiliza binario $rw\bar{x}=111$ y $rw- = 110$ y en la segunda se transforma a octal):

x	x	x	x	
	rw	7	Lectura , escritura y ejecución	
	rw-	6	Lectura , escritura	
	r-x	5	Lectura y ejecución	
	r-	4	Lectura	
	-wx	3	Escritura y ejecución	
	-w-	2	Escritura	
	-x	1	Ejecución	
	-	0	Sin permisos	
x	x	x	x	

De tal forma que podemos hacer:

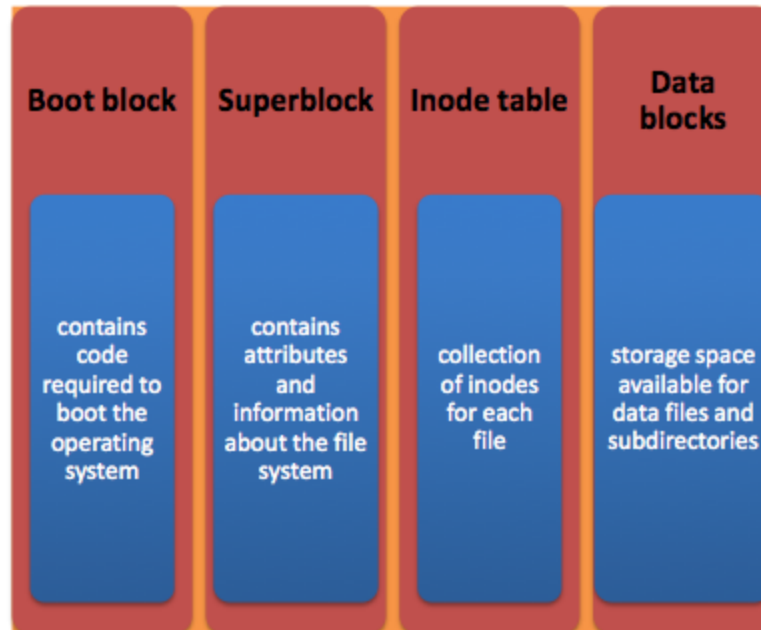
x		x		x
	chmod	u=rwx ,g=rwx ,o=r x		chmod 775
	chmod	u=rwx ,g=r x ,o=		chmod 760
	chmod	u=rw ,g=r ,o=r		chmod 644
	chmod	u=rw ,g=r ,o=		chmod 640
	chmod	u=rw ,go=		chmod 600
	chmod	u=rwx ,go=		chmod 700
x		x		x

3 Implementación del Sistema de Archivos

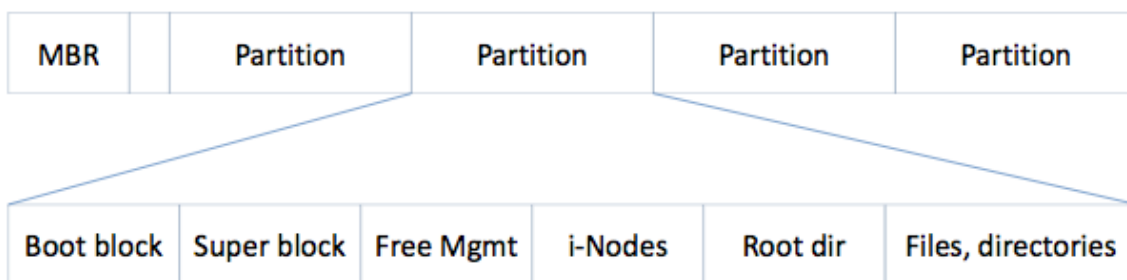
3.1 Sistemas de Archivos

- Gestionan el almacenamiento de datos en disco.
- La unidad de organización es el archivo → objeto de datos que ocupa espacio en disco.
- Organiza el espacio en disco → El disco llegar a ser un objeto contenedor de archivos.
- Gestiona:
 - Localización: registra dónde y como están almacenados los archivos.
 - Estructura: Archivos se organizan en directorios / carpetas.
 - Acceso: Permite la creación de archivos y operaciones R y WR.
 - Rendimiento: reduce operaciones I/O
 - Fiabilidad: puede recuperarse ante fallos.
 - Seguridad: privilegios de acceso

3.2 Unix Volume Management

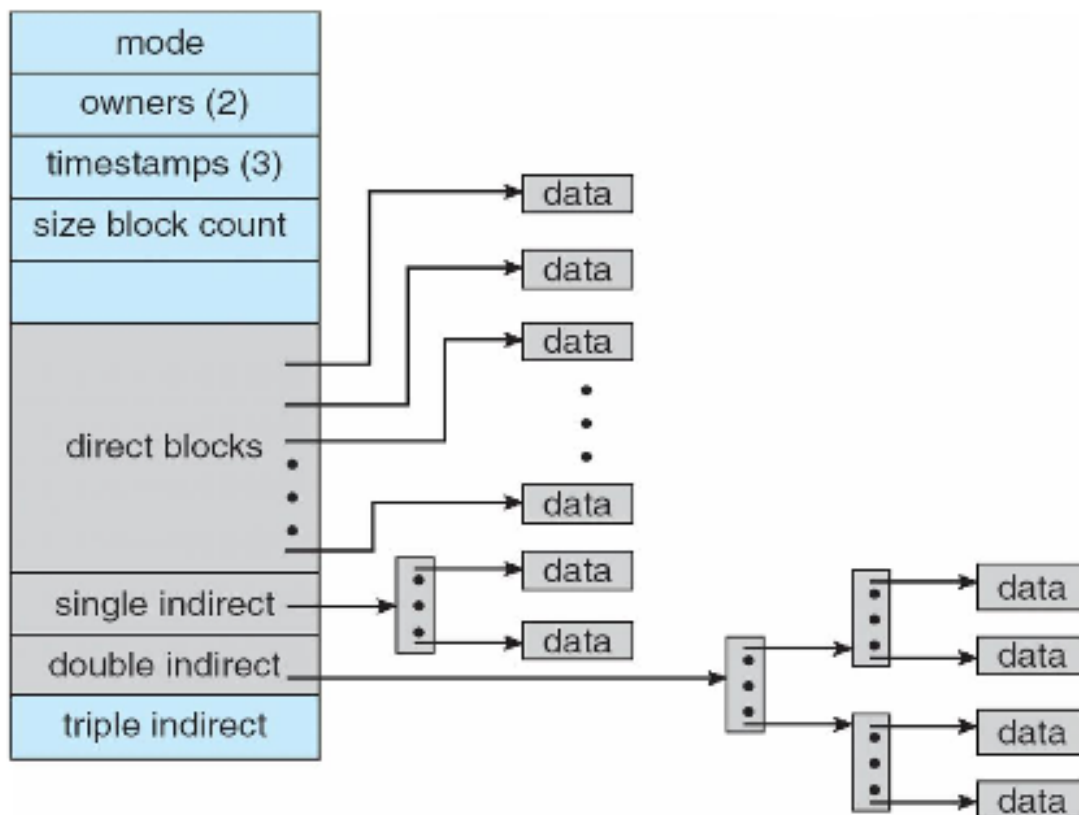


- El sistema de archivos de UNIX reside en un único disco lógico o partición.
- Se compone de:
 - Boot block: contiene el código requerido para iniciar el SO.
 - Superblock: contiene atributos e información sobre el sistema de archivos.
 - inode table: colección de inodos para cada archivo.
 - Data blocks: espacio de almacenamiento disponible para archivos y directorios.
- La capa de sistema de archivos y disco para UNIX funciona de la siguiente manera:
 - Master Boot Record (MBR): sector 0 del disco contiene el código de arranque y la tabla de particiones.
 - Arranque de Sistema: el programa contenido en el MBR carga el Boot Block de la partición activa o provee un menú para cargar otra.



3.3 Virtual File System (Linux)

- El VFS provee un interfaz común para diferentes sistemas de archivos. Además, añade un identificador único para los archivos a través de todo el disco incluyendo en otros Sistemas de Archivos de diferentes tipos → Esto lo hace bastante potente a la hora de poder utilizarse por otros Sistemas.
- El VFS de Linux está basado en cuatro objetos clave:
 - El objeto **inode**: que representa el archivo individual.
 - El objeto **archivo**: que representa un archivo abierto.
 - El objeto **superblock**: que representa el sistema de archivos.
 - El objeto **dentry**: representa una entrada a directorio.
- El **i-Node**:
 - Todos los tipos de archivos UNIX/Linux están gestionados por el SO mediante los i-Nodes.
 - Es una estructura de control (index node) que contiene la clave de información necesaria para el OS para un archivo particular (describe sus atributos y punteros a los bloques del disco donde está ubicado el archivo).
 - El i-Node es un índice los bloques de disco del archivo (un i-node por archivo).
 - Puede haber una serie de archivos para un sólo i-node (ligaduras o links, que son como alias).



4 Taller de Laboratorio

Este Taller de Laboratorio se realizará mediante la asignación de problemas y ejercicios prácticos vía la plataforma virtual Edmodo.

5 Bibliografía

References

- [1] William Stallings. *Operating Systems: Internals and Design Principles*. **Capítulos 11 y 12** Prentice Hall, 2012.
- [2] Abraham Silberschatz. *Operating Systems Concepts*. **Capítulos 10, 11 y 12** Wiley, 2012.