Índice

1. File systems

En la clase del 27 de Abril de 2020 se utilizó, en parte de la misma, las filminas que se encuentran en el documento **fsys-ppt.pdf** en el directorio *docs*.

1.1. Filminas

Estas filminas están extraídas, en gran parte, del capítulo *Files System Implementation* del libro *Operating Systems* de *Arpaci-Dusseau* que puene encontrar en el siguiente link: **File System Implementation**

1.2. Filmina 2: Unix: bloques de abstracción

Se muestran dos de las tres abstracciones de los sistemas operativos y, resumidamente, cómo se implementan en Linux; en referencia a este tema, se recuerda

que se vió con ejemplos en el caso de virtualización de proceso y memoria en la clase del 13 de abril y que ahora se reforzó en el caso de memoria en esta clase.

En la clase actual, se trata del Sistema de Archivo Virtual (en adelante **VFS** por *Virtual File System*) que implementa Unix/Linux y bajo el cual se abstrae todo el resto de los recursos de la computadora que no sean memoria o procesador; esto lleva a decir, no necesariamente en forma precisa, que en Unix *todo es archivo*.

Lo cierto es que todo lo que es entrada/salida (en adelante *I/O*) se trata de unificar bajo la abstracción extendida de *archivo* lo cual se ejemplifica en la filmina por tres grupos de *I/O* como son las terminales (bajo cuyo nombre se colocan todos los dispositivos en los cuales fundamentalmente se transacciona a nivel de bytes por lo cual se los conoce como *character oriented devices*), los sistemas de archivos los cuales se asientan sobre dispositivos donde la transferencia con el dispositivo se realiza por bloques de bytes (por lo cual se los conoce como *block oriented devices*) y los dispositivos de comunicaciones de redes.

Es interesante comprender que las transacciones que se realizan, entonces, desde el nivel de usuario a través del VFS, tienen disponibles muy pocos *system calls* cuales son *open*, *read*, *write*, y *close* (como se mostró usando algunos de ellos en el simple ejemplo de la clase anterior *15-io.c*).

Por lo tanto, esta sencilla abstracción (y cuando se dice *sencilla* es desde el punto de visto de cómo se entiende y usa esta abstracción y no necesariamente de su implementación) permite no hacer diferencias notables de código de cómo se tratan cada uno de los recursos y de su compatibilidad cuando de intercambiar datos entre ellos y con la memoria se trata.

1.3. Filmina 3: VFS y particiones

En esta filmina, se muestra la estructura jerárquica donde en la parte superior se encuentra la vista del usuario del sistema de archivos que se relaciona con el nivel siguiente, que es el VFS; en la parte inferior se ven los dispositivos orientados a bloque, los cuales se relacionan con los niveles superiores mediante las particiones que se realizan en ellos.

En cada una de las particiones, que se pueden usar como si fuesen discos separados, se puede colocar lo que se desee, entre otros, sistemas de archivos específicos, datos discrecionales en formatos variados, áreas propias de uso de sistemas operativos como áreas de *swap*, etc.

En este caso, estamos interesados primordialmente en *sistemas de archivos*. ¿Qué es un *sistema de archivo* ó *file system*?. No es otra cosa que una organización, en este caso en un disco duro (pero podría pensarse en una equivalente utilizando archivos en papel), de los datos que nos interesa resguardar y rescatar, organizados como *archivos de datos* así como también resguardar la forma de acceder a cada

uno de ellos mediante la organización simultánea de los *metadatos* (también conocido como *los datos de los datos*).

De acuerdo a la forma de organización, existen varios tipos conocidos, cada uno de ellos con ventajas y desventajas y realizados en distintas épocas y con distintos objetivos; teniendo en cuenta la razón de nuestro estudio, que está relacionado con Sistemas Operativos, usaremos una clasificación en relación a los mismos, a saber:

- Windows
- MacOS
- Linux
- BSD, Solaris, Unix
- Clustered File Systems

Nota: Para revisar más detalladamente estos conceptos, se sugiere ver **Understanding File Systems**.

1.4. Filminas 4 a 8

En estas filminas se muestra en forma muy simplificada como puede ser organizado un File System en disco del tipo de los que se utilizan en Unix (como por ejemplo, *ext2* y *ext3*).

Se muestran donde están los bloques de datos y como se organizan los metadatos, a través de los inodos y de los mapas de bits de ocupación de los bloques de datos y de los inodos.

1.5. Filmina 9

Se muestra un típico contenido de un inodo, donde se incluye el bloque de punteros a los bloques de datos que conforman el archivo.

1.6. Filmina 10

Se muestra como se accede, a través de la información anteriormente descripta, a los bloques que conforman el contenido de datos del archivo; en efecto, este acceso está dividido en 12 direcciones directas a los primeros 12 bloques de disco, una dirección indirecta a un bloque que tiene N direcciones directas, una dirección doble indirecta que tiene N direcciones a bloques que contiene cada uno a N direcciones directas y una dirección triple indirecta. (*Determine cual es la capacidad máxima de almacenamiento que podría tener un archivo si el tamaño de cada bloque de disco es de 4KByte y N es 1024*).

1.7. Filmina 11

Estadísticas típicas de archivos, en cantidad y tamaño, y de tamaño de directorios; explique porqué la organización mostrada en la filmina 10 en cuanto a forma de direccionamiento de bloques en un inodo toma en cuenta esta estadística.

---

2. Ejercitación:

Es de hacer notar que la ejercitación que sigue ha sido escrita y mostrada fundamentalmente para Linux; no todos los sistemas operativos procedentes de Unix son exactamente iguales cuando se entra en detalles de implementación interna y menos aún relacionados a los comandos que permiten ver y modificar comprtamientos internos.

Por lo tanto, se muestra con detalle el caso de Linux (especialmente porque el autor conoce un poco más internamente de él que de MacoS).

Se recomienda, entonces, al lector que vea lo escrito para Linux y en el segmento posterior donde se habla de MacOS, se indicn los comandos qequivalentes que se pueden usar en MacOS y las diferencias respecto de Linux.

De todas maneras, se incentiva que se realcen las mismas experiencias que en el caso de Linux, teniendo en cuenta las diferencias.

2.1. Sobre particiones

2.1.1. Linux

Ejemplificando sobre particiones, se pueden observar cuales y como son en un caso particular de sistema operativo e implementación en una computadora.

Para poder ver los dispositivos de bloque que se encuentran montados en un caso particular, (Computadora con un disco duro de 1Tbyte y un pen drive de 4 GByte, corriendo Linux 18.04) se puede realizar de las siguientes formas:

```
$ sudo fdisk -l | grep /dev/sd
```

```
Disk /dev/sda: 931,5 GiB, 1000204886016 bytes, 1953525168 sectors
/dev/sda1     2048    1050623    1048576  512M EFI System
/dev/sda2  1050624 1953523711 1952473088  931G Linux filesystem
Disk /dev/sdb: 3,7 GiB, 3932160000 bytes, 7680000 sectors
/dev/sdb1        1472 7679999 7678528  3,7G  b W95 FAT32
```

Se ve, claramente, que existen dos discos: el disco denominado /dev/sda de 1 TByte y el disco /dev/sdb de 4 GByte

Nota: No necesariamente reporta el tamaño exacto del disco ¿Piensa que hay un error o quizás valdría la pena investigar a qué se debe?

Este comando también reporta que en el primer caso existen dos particiones, a saber:

- /dev/sda1 de 512M (muy pequeña para el tamaño total del disco) que está rotulada como *EFI System*
- /dev/sda2 de casi la totalidad del disco, que está rotulada como *Linux filesystem*

En el segundo caso, existe una sola partición

- /dev/sdb1 de la totalidad del disco que está rotulada como *W95 FAT32*

Nota: sería interesante investigar sobre *W95 FAT32* pues parece ser un *file system*.

Existe otro comando que puede usarse, mediante los argumentos adecuados, para ver las particiones

```
$ df -hT | grep /dev/sd

/dev/sda2      ext4      916G  148G  722G  17% /
/dev/sda1      vfat      511M   13M  499M   3% /boot/efi
/dev/sdb1      vfat      3,7G  3,4G  342M  91% /media/tedmar/BLUE PEN 2
```

Es interesante ver que este comando sólo reporta las particiones con otra información, como ser que encuentra *file systems* en todos ellos y da el tipo de los mismos.

Además, da la partición y el grado de ocupación asi como la proporción de ocupación de la partición

Existe también al final de cada línea una especie de identificación de directorio; en efecto, esto es propio de Linux, ya que una partición de disco para poder verse posteriormente desde VFS debe ser *montada* sobre un directorio existente de VFS.

Si bien no se explicó en clase y tampoco se lo hará aquí, se sugiere ver como se *monta* (*mount*) y *desmonta* (*umont*) un dispositivo en Unix/Linux. Para practicar, no use partción alguna de un disco duro y pruebe con un pen drive en USB. (Obviamente, puede consultar el manual en línea o buscar una página de Web suficientemente clara).

Con respecto a los comandos utilizados, a saber **fdisk** y **df** fíjese que hacen y cómo se usan a través del manual en línea o también buscando alguna página de Web.

También puede investigar una herramienta gráfica llamada *gparted* que permite ver y crear particiones en forma muy interactiva; investigue sobre ella.

Recuerde también que así como *fdisk* necesita permiso de superusuario, también *gparted* lo necesita, a través de invocar mediante *sudo*: ponga especial cuidado de lo que hace en estos casos pues recuerde que el superusuario tiene habilitado todo en la computadora a través del sistema operativo.

2.1.2. MacOS

En este caso, se pueden visualizar los discos y particiones mediante el aplicativo *Disk Utility.app*, que es de tipo GUI y puede invocarlo mediante *cmd-space*; puede consultar la página de Web: **How to show all drive devices with Disk Utility**.

También puede ver los siguientes comandos a nivel de CLI: *fdisk* (que no es equivalente al de Linux), *pdisk* y *gpt*.

Lea todo lo relacionado a Linux en el párrafo anterior y trate de rehacer en este sistema operativo lo especificado, cambiando adecuadamente los comandos.

2.2. Directorios específicos ubicados en /

2.2.1. Directorio /dev

2.2.1.1. Linux

Hemos estado viendo, cuando vimos particiones, que los discos aparecían como, por ejemplo, /dev/sdb1

Obviamente, existe un directorio /dev y si hacemos un listado del directorio como

```
$ ls -l /dev | head -15

total 0
crw-------  1 root   root     10,    55 abr 28 09:32 acpi_thermal_rel
crw-r--r--  1 root   root     10,   235 abr 28 09:32 autofs
drwxr-xr-x  2 root   root          1040 abr 30 14:03 block
drwxr-xr-x  2 root   root            80 abr 30 14:03 bsg
crw-------  1 root   root     10,   234 abr 28 09:32 btrfs-control
drwxr-xr-x  3 root   root            60 abr 28 09:32 bus
drwxr-xr-x  2 root   root          4700 abr 30 16:01 char
```

```
crw-------  1 root   root      5,     1 abr 28 09:32 console
lrwxrwxrwx  1 root   root            11 abr 28 09:32 core -> /proc/kcore
drwxr-xr-x  2 root   root            60 abr 28 09:32 cpu
crw-------  1 root   root     10,    59 abr 28 09:32 cpu_dma_latency
crw-------  1 root   root     10,   203 abr 28 09:32 cuse
drwxr-xr-x  8 root   root           160 abr 30 14:03 disk
drwxr-xr-x  3 root   root           140 abr 28 09:32 dri
```

Vemos, entonces, que se muestran directorios y algunos archivos, los cuales no son regulares, pues en la primera letra de los permisos tienen una **c** (más adelante hablaremos del archivo **core** que tiene una **l** delante).

Vamos a hacer otro listado:

```
$ ls -l /dev/sd* | head -15

brw-rw---- 1 root disk 8,  0 abr 28 09:32 /dev/sda
brw-rw---- 1 root disk 8,  1 abr 28 09:32 /dev/sda1
brw-rw---- 1 root disk 8,  2 abr 28 09:32 /dev/sda2
brw-rw---- 1 root disk 8, 16 abr 30 16:00 /dev/sdb
brw-rw---- 1 root disk 8, 17 abr 30 14:03 /dev/sdb1
```

Casualmente, estos son los discos y sus particiones que descubrimos anteriormente; en este caso, todos poseen como primera letra en los permisos la letra **b**.

Básicamente, los archivos que se encuentran en /dev se los denomina *archivos especiales*.

Estos archivos se los llama *archivos de dispositivos* o, en inglés *device files* y vemos que existen en dos tipos: dispositivos orientados a *bloques* o dispositivos orientados a *caracter* (obviamente, la primera letra indica que son archivos especiales de dispositivos y la letra coincide con el tipo).

El archivo cuya primera letra del campo de permisos es una **l** no es propio solamente de este directorio, sino que se lo puede encontrar en otros directorios: se trata de lo que se llama una *liga simbólica* (también llamada *soft link*); más adelante volveremos sobre este tema.

Por formar parte de un sistema de archivos, obviamente se podrían realizar las operaciones básicas sobre los archivos de dispositivos como *open*, *read*, *write* y *close*: veamos si eso es posible; vamos a utilizar el archivo de dispositivo **/dev/tty** de la siguiente forma:

```
$ cat < /dev/tty > file
Hola, este es un ejemplo de imprimir sobre stdout
lo que ingresa desde el arhivo de dispositivo
/dev/tty; a su vez, la salida stdout de cat se
redirecciona al archivo file.
```

```
$ cat file > /dev/tty
Hola, este es un ejemplo de imprimir sobre stdout
lo que ingresa desde el arhivo de dispositivo
/dev/tty; a su vez, la salida stdout de cat se
redirecciona al archivo file.
```

Vemos, entonces, dos comandos: el primero es simplemente el comando *cat* que lista el archivo de entrada sobre *stdout*; de todas maneras, se puede redirigir la entrada (*stdin*) desde el archivo de dispositivo /dev/tty y también redirigir la salida (*stdout*) al archivo de nombre file. Como el dispositivo /dev/tty coincide siempre con la terminal del usuario, entonces lo que hará este comando es tomar la entrada desde consola escrita a través del teclado y copiar esa entrada al archivo de nombre *file*. ¿Cuando se considerará terminado el ingreso desde /dev/tty?. Sencillamente cuando se encuentre *end of file* lo cual desde el teclado se señaliza con control-D.

Una vez terminado el primer comando, se envía otro comando *cat* tomando la entrada desde el archivo generado en el anterior comando y redireccionando la salida *stdout* ahora a la entrada del archivo especial /dev/tty, con lo cual se muestra en la salida standard de este último.

2.2.1.2. MacOS

Prácticamente, todo lo visto aquí es válido para MacOS; seguramente va a encontrar algunas diferencias con el caso de Linux.

Una de ellas es que los discos no se nombran de la misma manera, lo cual ya pudo verlo en el párrafo anterior sobre particiones cuando invocó a *Disk Utility.app*.

Por lo tanto, haga la ejercitación teniendo en cuenta estas diferencias.

2.2.2. Directorio /proc

2.2.2.1. Linux

Si lista este directorio con el comando

```
$ ls /proc
```

Se va a encontrar con una gran cantidad de subdirectorios y archivos; si determina cual es el tamaño de archivos y directorios *todos ellos son de tamaño 0* lo cual si es extraño; pues bien, este directorio /proc es un *pseudo filesystem* que permite el acceso a ver y modificar datos del *kernel*.

Observemos alguno de los archivos que lo conforman, como es el caso de *meminfo* haciendo el siguiente comando

```
$ head -15 /proc/meminfo

MemTotal:       16259840 kB
MemFree:          802724 kB
MemAvailable:    4347708 kB
Buffers:          756112 kB
Cached:          3924272 kB
SwapCached:         1596 kB
Active:         11135584 kB
Inactive:        3126448 kB
Active(anon):    9403500 kB
Inactive(anon):  1418960 kB
Active(file):    1732084 kB
Inactive(file):  1707488 kB
Unevictable:         584 kB
Mlocked:              64 kB
SwapTotal:       2097148 kB
```

Se ve que, aparentemente en forma mágica, un archivo reportado de tamaño 0, presenta una gran cantidad de información; esto es porque cada vez que se lee uno de estos archivos, se está pidiendo información de variables y actividad del *kernel* la cual se busca y reporta en el momento.

Otro archivo interesante es *cpuinfo*

```
$ head -15 /proc/cpuinfo

processor   : 0
vendor_id   : GenuineIntel
cpu family  : 6
model       : 158
model name  : Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz
stepping    : 9
microcode   : 0xca
cpu MHz     : 1858.029
cache size  : 6144 KB
physical id : 0
siblings    : 8
core id     : 0
cpu cores   : 4
apicid      : 0
initial apicid  : 0
```

También se puede cambiar el comportamiento del *kernel* escribiendo en algunos archivos, lo cual por ahora no se aconseja tratar de hacerlo sin estar muy seguro.

Más sobre este admirable directorio, ver **proc pseudo filesystem**

2.2.2.2. MacOS

En este caso, Linux y MacOS son totalmente diferentes, ya que no existe un directorio */proc*.

El caso de */proc/meminfo* puede ser reemplazado por **vm_stat**.

/proc/cpuinfo puede ser reemplazado por **sysctl -a | grep machdep-cpu | less**.