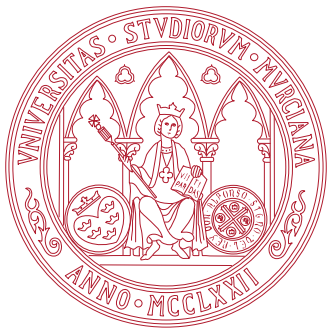

Universidad de Murcia

Facultad de Informática

Departamento de Ingeniería y Tecnología de Computadores

Área de Arquitectura y Tecnología de Computadores

Prácticas de

Introducción a los Sistemas Operativos

2º de Grado en Ingeniería Informática

Boletín de Prácticas 6 – Gestión de discos

Curso 2021/2022

Índice

1. Introducción

En la actualidad, podemos encontrar diversos tipos de dispositivos para almacenar información de forma no volátil. De ellos, los discos duros siguen siendo los más utilizados, al ofrecer grandes capacidades y un bajo coste por gigabyte, si bien los discos SSD cada vez son más populares por su mayor rendimiento y porque su precio se ha reducido considerablemente en los últimos años. Los discos duros y SSD se enfrentan, sin embargo, a dos grandes problemas: rendimiento y fiabilidad.

Respecto al rendimiento, aunque es cierto que los discos duros han evolucionado a lo largo de los años y que hoy en día son mucho más rápidos que hace unas décadas, también es cierto que sus mejoras no se han producido al mismo ritmo ni han sido tan grandes como las que se han dado en otros componentes, como procesadores, memoria, etc. Esto hace que estos dispositivos sean uno de los principales cuellos de botella que podamos encontrar en un sistema informático. En el caso de los discos SSD, su rendimiento es mucho mayor que el de un disco duro tradicional, pero, aun así, no es comparable al de esos otros componentes.

La fiabilidad de la información almacenada en los discos es otro aspecto importante a tener en cuenta. Es fácil entender que si un procesador o un módulo de memoria falla, la solución es rápida: basta con sustituir el componente que ha fallado. Sin embargo, si un disco falla, se puede producir una pérdida de información que no es recuperable sustituyendo el disco defectuoso por otro. Además, otro problema de los discos es que algunos tienen una capacidad tan alta que la probabilidad de que existan o de que surjan sectores o zonas defectuosas es cada vez mayor.

En este boletín vamos a describir primero cómo preparar un disco para su uso creándole particiones. Después, veremos cómo *virtualizar* el almacenamiento para, combinando varios discos, «crear» nuevos discos que sean más rápidos y/o más fiables que un solo disco, puedan cambiar de tamaño de forma sencilla, etc. Finalmente, explicaremos cómo usar simples ficheros regulares como si fueran discos. Esto nos permitirá experimentar sin poner en peligro los datos de nuestro sistema.

2. Objetivos

Al terminar este boletín, el alumno debe ser capaz de:

- Describir qué son los dispositivos de bloques y los ficheros especiales de bloques que los representan.
- Preparar un disco usando particiones DOS, sabiendo distinguir entre partición primaria, extendida y lógica.
- Preparar un disco usando particiones GPT y describir el uso de los GUIDs.
- Listar los dispositivos de bloques disponibles y saber identificarlos de distintas formas.
- Entender qué es un sistema LVM y qué ventajas ofrece.
- Conocer los principales elementos que componen un sistema LVM: grupos de volúmenes, volúmenes físicos y lógicos, y extensiones físicas y lógicas.
- Administrar dispositivos LVM en Linux: crear un grupo de volúmenes, añadir y eliminar volúmenes físicos, crear volúmenes lógicos, consultar el estado de los distintos elementos, cambiar el tamaño de un volumen lógico, etc.
- Crear distintos tipos de volúmenes lógicos: lineal, repartido, reflejado y raid5.
- Obtener información de cómo se corresponden las extensiones lógicas con las físicas y saber interpretar dicha información.
- Gestionar dispositivos *loop*.

3. Ficheros y órdenes utilizados

En este boletín, nos centraremos en los siguientes ficheros y directorios:

- `/dev`
- `/dev/mapper`
- `/dev/<grupo_de_volúmenes>`

También veremos, entre otras, las siguientes órdenes:

- `fdisk`
- `lsblk`
- `pvcreate`
- `pvdisplay`
- `pvremove`
- `pvmove`
- `vgcreate`
- `vgdisplay`
- `vgextend`
- `vgreduce`
- `vgremove`
- `lvcreate`
- `lvdisplay`
- `lvextend`
- `lvreduce`
- `lvremove`
- `dd`
- `losetup`

4. Dispositivos y ficheros especiales de bloques

Como veremos en teoría, un *dispositivo de bloques* es un dispositivo que almacena información en bloques de tamaño fijo, cada uno con su propia dirección, lo que permite leer de o escribir en un bloque de forma independiente de los demás. Hoy en día, los principales dispositivos de bloques son los discos duros, discos SSD y memorias USB; otros dispositivos de bloques son las unidades de CD/DVD, memorias flash NVMe, etc. En este boletín nos centraremos en los discos.

En Unix, y en Linux en particular, se utilizan *ficheros especiales* para representar a diferentes elementos del sistema. Dentro de estos ficheros, tenemos los *ficheros especiales de bloques* que, como su propio nombre indica, sirven para dar nombre a los dispositivos de bloques y, en especial, a los discos. A través de estos ficheros seremos capaces de realizar ciertas operaciones sobre este tipo de dispositivos.

Los ficheros especiales de bloques se distinguen del resto porque sus permisos comienzan por la letra «b». A continuación, mostramos algunos ficheros de este tipo que, habitualmente, se encuentran en el directorio `/dev`:

```
[alumno@localhost ~]$ ls -l /dev/sd*
brw-rw----. 1 root disk 8,  0 feb 23 15:54 /dev/sda
brw-rw----. 1 root disk 8,  1 feb 23 15:54 /dev/sda1
brw-rw----. 1 root disk 8,  2 feb 23 15:54 /dev/sda2
brw-rw----. 1 root disk 8, 16 feb 23 15:54 /dev/sdb
brw-rw----. 1 root disk 8, 32 feb 23 15:54 /dev/sdc
brw-rw----. 1 root disk 8, 48 feb 23 15:54 /dev/sdd
```

En este ejemplo podemos encontrar los ficheros especiales de bloques que representan a los cuatro discos duros de nuestra máquina virtual (`sda`, `sdb`, `sdc` y `sdd`) y a las dos particiones que hay en el primer disco (`sda1` y `sda2`). Los discos `sdb`, `sdc` y `sdd` no tienen particiones todavía.

5. Particiones

Aunque es posible usar un disco sin «particionarlo», lo normal es que un disco tenga una o más particiones que abarquen todo el espacio de almacenamiento disponible. Una partición es un número de sectores consecutivos dentro de un disco. Al crear una partición, lo que hacemos es crear un dispositivo de bloques dentro de otro ya existente. Las particiones se usan por diversos motivos:

- Si se desea instalar varios sistemas operativos en un mismo computador, será necesario tener una partición para cada uno de ellos.

- Algunos sistemas de ficheros tienen un tamaño máximo menor que el del disco que usan. Para aprovechar el disco, hay que crear varias particiones que lo ocupen todo, pero que sean lo suficientemente pequeñas como para que el sistema de ficheros las use completamente.
- Muchos ordenadores (sobre todo portátiles) mantienen en una partición una copia del SO lista para instalar por si hubiera problemas con la instalación principal.
- Los sistemas UNIX usan muchas veces diferentes particiones para almacenar el contenido de diversos directorios y así minimizar el impacto de actualizaciones, la posible corrupción de un sistema de ficheros, etc. En estos sistemas es habitual tener:
  - Una partición para la instalación del sistema operativo.
  - Otra partición para almacenar los ficheros de los usuarios.
  - Una tercera partición para la zona de intercambio o *swap*, que almacena las páginas que no caben en memoria RAM.

Es importante darse cuenta de que las particiones solo existen para nuestro sistema operativo. El disco, como tal, no conoce la existencia de las particiones. Las particiones son, por tanto, componentes lógicos que el sistema operativo gestiona mediante una estructura de datos que almacena en el propio disco. Esta estructura de datos puede construirse de distintas maneras, lo que da lugar a distintos tipos de particionamiento. Los siguientes apartados describen los particionamientos DOS y GPT.

5.1. Particionamiento DOS

Históricamente, en los PC compatibles se ha utilizado el formato de particiones original de MS-DOS, de ahí que a este particionamiento se le llame DOS[1].

Este particionamiento se diseñó inicialmente para gestionar hasta cuatro particiones en un disco. Sin embargo, el tamaño de los discos duros creció y hubo que buscar una solución a este límite en el número de particiones. La solución consistió en dotar de una naturaleza especial a una de esas cuatro particiones, pasándose a denominar *partición extendida*, mientras que a las restantes particiones iniciales se les denominó *particiones primarias*. Una partición extendida es el único tipo de partición que no soporta un sistema de ficheros directamente. Dentro de la partición extendida pueden crearse las *particiones lógicas* (también llamadas *secundarias*) que son a las que se les puede dar formato y contener, por tanto, un sistema de ficheros.

Podemos obtener la información general sobre las particiones existentes en un dispositivo mediante la orden `fdisk` de esta manera:

```
# fdisk -l /dev/sdb
Disco /dev/sdb: 1 GiB, 1073741824 bytes, 2097152 sectores
Unidades: sectores de 1 * 512 = 512 bytes
Tamaño de sector (lógico/físico): 512 bytes / 512 bytes
Tamaño de E/S (mínimo/óptimo): 512 bytes / 512 bytes
Tipo de etiqueta de disco: dos
Identificador del disco: 0x6eb6c438

Disposit.  Inicio Comienzo   Final Sectores Tamaño Id Tipo
/dev/sdb1             2048  411647   409600   200M 83 Linux
/dev/sdb2           411648 2097151  1685504   823M  5 Extendida
/dev/sdb5           413696 2097151  1683456   822M 83 Linux
```

En este ejemplo, apreciamos que el dispositivo `/dev/sdb` tiene una partición primaria, `/dev/sdb1`, y una extendida, `/dev/sdb2`. A su vez, dentro de la extendida, existe la partición lógica `/dev/sdb5`.

[1] También se le llama MBR (Master Boot Record) porque se denomina así al primer sector (el 0) del disco. Este sector almacena (total o parcialmente) la tabla de particiones del disco

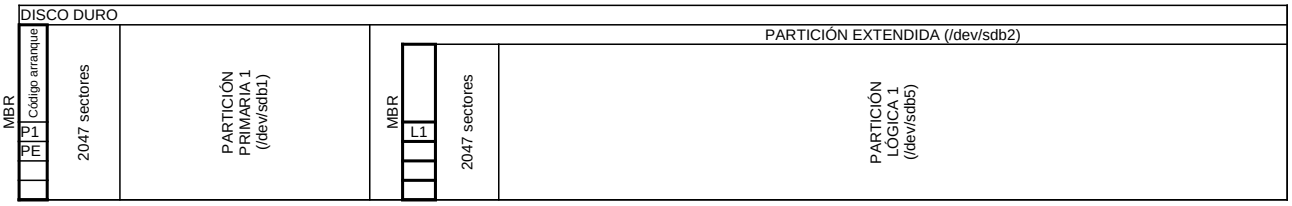

Figura 1: Disco con particionamiento DOS. Corresponde con la salida de `fdisk` mostrada en el texto. Vemos que la partición extendida se trata como un nuevo disco y, por tanto, tiene un MBR con su correspondiente tabla de particiones.

En la figura 1 se puede ver una versión simplificada de cómo se distribuirían estas particiones por el disco.

En la salida de `fdisk` también podemos observar que la primera partición se encuentra a partir del sector 2 048. En la actualidad, este hueco se usa comúnmente para contener un gestor de arranque. Los sectores se numeran desde 0, siendo 0 el primer sector del disco. Al número que recibe cada sector se le denomina LBA (Logical Block Addressing) y constituye su dirección.

Por desgracia este esquema tradicional de particionamiento tiene el inconveniente de que el número de sector de un disco debe caber en 32 bits. Puesto que el sector tiene típicamente un tamaño de 512 bytes, esto nos limita a discos de 2 TiB como máximo. En el caso de tener un disco más grande, nuestras particiones no podrían superar ese límite, dejando el resto del disco inutilizable. Los nuevos discos con tamaño de sector de 4 KiB no arreglan el problema, solo lo retrasan un poco, ya que nos permiten usar hasta 16 TiB. Pero, además, los sectores de 4 KiB no están soportados por todos los sistemas, así que habitualmente el disco los usa internamente y sigue mostrando una interfaz con sectores de 512 bytes, por lo que el problema sigue siendo el mismo.

A continuación mostramos cómo se ha usado la orden `fdisk` para crear las particiones anteriores; en negrita aparecen los valores introducidos por teclado:

```
# fdisk /dev/sdb

Bienvenido a fdisk (util-linux 2.32).
Los cambios solo permanecerán en la memoria, hasta que decida escribirlos.
Tenga cuidado antes de utilizar la orden de escritura.


Orden (m para obtener ayuda): n
Tipo de partición
   p   primaria (0 primaria(s), 0 extendida(s), 4 libre(s))
   e   extendida (contenedor para particiones lógicas)
Seleccionar (valor predeterminado p): <return>

Se está utilizando la respuesta predeterminada p.
Número de partición (1-4, valor predeterminado 1): <return>
Primer sector (2048-2097151, valor predeterminado 2048): <return>
Último sector, +sectores o +tamañoK,M,G,T,P (2048-2097151, valor
predeterminado 2097151): +200M

Crea una nueva partición 1 de tipo 'Linux' y de tamaño 200 MiB.

Orden (m para obtener ayuda): n
Tipo de partición
   p   primaria (1 primaria(s), 0 extendida(s), 3 libre(s))
   e   extendida (contenedor para particiones lógicas)
Seleccionar (valor predeterminado p): e
Número de partición (2-4, valor predeterminado 2): <return>
Primer sector (411648-2097151, valor predeterminado 411648): <return>
Último sector, +sectores o +tamañoK,M,G,T,P (411648-2097151, valor
predeterminado 2097151): <return>
```

```
Crea una nueva partición 2 de tipo 'Extended' y de tamaño 823 MiB.

Orden (m para obtener ayuda): n
Se está utilizando todo el espacio para particiones primarias.
Se añade la partición lógica 5
Primer sector (413696-2097151, valor predeterminado 413696): <return>
Último sector, +sectores o +tamañoK,M,G,T,P (413696-2097151, valor
predeterminado 2097151): <return>

Crea una nueva partición 5 de tipo 'Linux' y de tamaño 822 MiB.

Orden (m para obtener ayuda): w
Se ha modificado la tabla de particiones.
Llamando a ioctl() para volver a leer la tabla de particiones.
Se están sincronizando los discos.
```

Como podemos ver, en muchos casos hemos pulsado la tecla «return» para aceptar el valor predeterminado. Además de la orden «`n`» para crear una nueva partición y la orden «`w`» para guardar en disco la nueva tabla de particiones, también podemos usar «`p`» para ver la tabla de particiones, «`d`» para borrar una partición o «`q`» para salir sin guardar los cambios. La orden «`m`» nos muestra una ayuda con todas las posibilidades.

Hay varios detalles que merece la pena destacar. El primero es que hemos indicado el tamaño de la primera partición introduciendo `+200M` y no un número de sector. Estas expresiones nos permiten indicar de una forma sencilla el tamaño de una partición sin tener que indicar su último sector ni hacer cálculos de tamaño a mano.

Otro detalle es que las particiones lógicas comienzan a partir de 5. Esto es así porque los números del 1 al 4 se reservan para las particiones primarias y extendidas.

Un último detalle a destacar es que las particiones tienen tipo. El tipo de una partición indica qué va a contener o para qué se va a usar. En nuestro ejemplo, para las particiones 1 y 5, el tipo que se ha seleccionado por defecto es «Linux», cuyo código en hexadecimal es 83. Esto indica que estas particiones van a contener un sistema de ficheros de Linux, *pero no que lo contengan ya*. De hecho, será necesario formatear cada partición antes de usarla, como veremos en el siguiente boletín. Es más, podremos formatear una partición de tipo «Linux» con cualquier sistema de ficheros, aunque no sea nativo de Linux (como VFAT). Por lo tanto, debemos entender el tipo de una partición más como una etiqueta que como una restricción, aunque, para evitar posibles problemas, es conveniente que el tipo de una partición siempre concuerde con aquello que va a contener.

Otros tipos interesantes para una partición son «Linux swap» (82), cuando la partición se va a usar para espacio de intercambio de la memoria virtual, y «Linux LVM» (8e), cuando se va a usar como volumen físico de un grupo de volúmenes (ver sección 7). El tipo de una partición se puede cambiar con la orden «`t`» de `fdisk`.

5.2. Particionamiento GPT

El esquema GPT (GUID Partition Table) permite discos mucho mayores que su predecesor al usar LBAs de 64 bits para almacenar el sector de comienzo y la longitud de las particiones, por lo que, con sectores de 512 bytes, es posible usar discos de hasta 8 ZiB. También admite un número prácticamente ilimitado de particiones, aunque los sistemas operativos suelen limitarlo a 128 para mantener la tabla de particiones en un tamaño razonable de 16 KiB.

En GPT, se utiliza un GUID (Globally Unique Identifier) para identificar unívocamente un disco o partición. Un GUID consiste en un número de 16 bytes (32 dígitos hexadecimales) dividido en 5 campos (p.e. 3F2504E0-4F89-41D3-9A0C-0305E82C3301) que se asigna a un recurso determinado y se utiliza para referenciarlo. Puede ser pseudoaleatorio en todos o algunos de sus campos. La idea es garantizar que sea altamente improbable tener dos recursos con el mismo identificador en el mismo sistema, aún cuando un recurso se mueva de un sistema a otro.

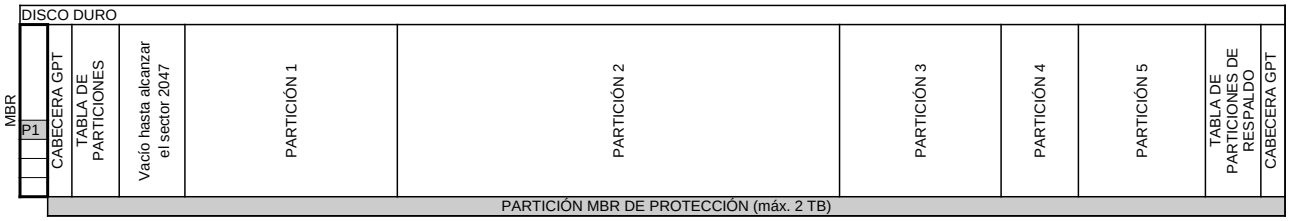

Figura 2: Disco configurado con el esquema GPT.

GPT también utiliza un GUID para identificar el tipo de una partición, aunque, ahora, el GUID no es pseudoaleatorio, sino que tiene un valor predeterminado y conocido. Así, por ejemplo, una partición básica de datos para Windows tiene el GUID EBD0A0A2-B9E5-4433-87C0-68B6B72699C7, mientras que si la partición va a ser usada para los datos de un sistema de ficheros de Linux, su GUID será 0FC63DAF-8483-4772-8E79-3D69D8477DE4. No existe una regla para deducir estos valores, que habitualmente son decididos por cada fabricante/sistema operativo.

El esquema GPT almacena la tabla de particiones en el sector 1 y sucesivos. También guarda una suma de verificación de la integridad de la tabla (CRC32) y una copia de respaldo en los últimos sectores del disco. Por compatibilidad con aquellos sistemas que no entienden GPT, el sector 0 almacena una tabla de particiones DOS con una única partición de tipo `0xEE` (tipo GPT, denominada también partición MBR de protección), que será tan grande como sea posible para intentar cubrir todo el disco. De esta manera, los sistemas operativos que no manejen GPT verán una partición de tipo desconocido y no tocarán el disco. La figura 2 ilustra un disco con particionamiento GPT.

La orden `fdisk` también permite crear una tabla de particiones GPT. Basta con usar la orden «`g`» antes de crear las particiones para indicarlo. Las órdenes de `fdisk` para crear particiones, borrarlas, ver la tabla de particiones, etc., son las mismas. La principal diferencia es que desaparece la distinción entre particiones primarias, extendidas y lógicas, siendo todas las particiones iguales y simplemente eso, particiones.

6. Listar dispositivos de bloques

Podemos obtener una lista de los dispositivos de bloques disponibles y de sus particiones con la orden `lsblk`. Sin argumentos, se muestra un listado en forma de árbol en la que los hijos de un nodo son los dispositivos que contiene, como se ven en el siguiente ejemplo:

```
# lsblk
NAME             MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda                8:0    0   15G  0 disk
├─sda1             8:1    0    1G  0 part /boot
└─sda2             8:2    0   14G  0 part
  ├─fedora-root 253:0    0 12,5G  0 lvm  /
  └─fedora-swap 253:1    0  1,5G  0 lvm  [SWAP]
sdb                8:16   0    1G  0 disk
sdc                8:32   0    1G  0 disk
sdd                8:48   0    1G  0 disk
```

Entre otra información, se nos muestra el nombre del dispositivo de bloques (columna NAME), su tamaño (columna SIZE) y su tipo (columna TYPE): disco, partición, volumen lógico (ver siguiente apartado), etc. El significado de las otras columnas lo veremos en boletines posteriores.

Una opción útil de esta orden es `-p`, que muestra la ruta absoluta del fichero especial de bloque que representa a cada dispositivo:

```
# lsblk -p
NAME                          MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
/dev/sda                        8:0    0   15G  0 disk
├─/dev/sda1                     8:1    0    1G  0 part /boot
```

```
└─/dev/sda2                    8:2    0   14G  0 part
  ├─/dev/mapper/fedora-root 253:0    0 12,5G  0 lvm  /
  └─/dev/mapper/fedora-swap 253:1    0  1,5G  0 lvm  [SWAP]
/dev/sdb                       8:16   0    1G  0 disk
/dev/sdc                       8:32   0    1G  0 disk
/dev/sdd                       8:48   0    1G  0 disk
```

Una última opción interesante es `-f`, que muestra información del sistema de ficheros contenido en cada dispositivo de bloques, si es el caso:

```
# lsblk -f
NAME            FSTYPE      LABEL UUID                                   MOUNTPOINT
sda
├─sda1          ext4              58a955b2-9209-4d41-bb53-d8fe87178cf0   /boot
└─sda2          LVM2_member       AAHTs2-wpvL-HXf5-Dv6W-ZIAA-Vksh-3YI7HP
  ├─fedora-root ext4              4b0782d8-de80-4833-8baf-7982a07de180   /
  └─fedora-swap swap              1c0c1afd-e3c6-4f15-87b8-03b6f19a282b   [SWAP]
sdb
sdc
sdd
```

Como vemos, aparecen nuevas columnas, como el tipo del sistema de ficheros (columna FSTYPE), su etiqueta (columna LABEL) y su identificador (columna UUID). También aparece el punto de montaje (columna MOUNTPOINT). En el siguiente boletín veremos el significado y uso de todos estos campos.

7. Sistemas LVM

Es deseable que el almacenamiento en disco sea flexible para poder evolucionar; con el tiempo, es habitual que sea necesario añadir nuevos discos para ampliar la capacidad de almacenamiento, reemplazar discos pequeños por otros grandes (o discos lentos por otros más rápidos), sustituir discos defectuosos, crear nuevas particiones, etc. Si usamos los discos y sus particiones directamente, estas tareas supondrán copiar «a mano» información de unas particiones a otras y, en algunos casos, reiniciar el sistema o arrancarlo con un disco de apoyo. Todo esto lleva tiempo, es propenso a errores y puede dar lugar a paradas largas del sistema.

Para solucionar los problemas mencionados, surgen los *gestores de volúmenes lógicos* o sistemas LVM (*Logical Volume Manager*), que «virtualizan» el almacenamiento para crear *discos lógicos* y proporcionar métodos de asignación de espacio en disco más flexibles que las simples particiones. En concreto, un gestor de volúmenes permite combinar particiones de distintos discos para crear particiones lógicas o virtuales que los administradores pueden mover o redimensionar mientras el sistema se sigue usando. Muchos sistemas LVM, como el de Linux, también permiten crear dispositivos lógicos de bloques con más capacidad, mayor rendimiento y/o mayor fiabilidad que un simple disco.

La mayoría de sistemas LVM comparten los mismos principios básicos de diseño. Aquí describiremos esos principios, pero basándonos en la implementación actual del sistema LVM proporcionado por Linux.

7.1. Arquitectura de un sistema LVM

Un gestor de volúmenes parte de *volúmenes físicos* (*physical volumes* o PVs), que pueden ser discos enteros, particiones u otros dispositivos de bloques cualesquiera. Cada volumen físico está dividido en bloques llamados *extensiones físicas* (*physical extents* o PEs), todos del mismo tamaño. Por omisión, este tamaño es de 4 MiB en Linux.

Podemos convertir cualquier dispositivo de bloques (disco, partición, ...) en un volumen físico, para que pueda ser usado en un sistema LVM, mediante la orden `pvcreate`. Así, suponiendo que los tres discos de nuestra máquina virtual tienen una única partición que ocupa todo el disco y que esta es de tipo «Linux LVM», podemos convertir dichas particiones en volúmenes físicos ejecutando:

```
# pvcreate /dev/sd[bcd]1
```

Una vez creados, podemos consultar las características de cada volumen físico mediante la orden `pvdisplay`, por ejemplo:

```
# pvdisplay /dev/sdb1
```

Si no indicamos un volumen físico como parámetro, nos aparecerá la información de todos los disponibles.

Los volúmenes físicos se agrupan en *grupos de volúmenes* (*volume groups* o VGs), de tal manera que las extensiones físicas de todos ellos conforman el almacén o repositorio de extensiones físicas del grupo de volúmenes. Por ejemplo, podemos crear un grupo de volúmenes llamado `vg-iso` a partir de los tres volúmenes físicos anteriores de la siguiente manera[2]:

```
# vgcreate vg-iso /dev/sd[bcd]1
```

Tras crear el grupo de volúmenes, podemos usar `vgdisplay` para conocer sus características:

```
# vgdisplay vg-iso
  --- Volume group ---
  VG Name               vg-iso
  ....
  VG Size               <2,99 GiB
  PE Size               4,00 MiB
  Total PE              765
  Alloc PE / Size       0 / 0
  Free  PE / Size       765 / <2,99 GiB
  ....
```

Vemos que nuestro grupo de volúmenes dispone de 765 extensiones físicas (de 4 MiB cada una, es decir, 2,99 GiBs) que están todas libres. De nuevo, si no ponemos el nombre del grupo de volúmenes como parámetro, se nos mostrará información de todos los grupos de volúmenes existentes. También podemos usar la opción `-v|--verbose` de esta orden para ver qué volúmenes físicos forman parte de cada grupo de volúmenes.

Para poder usar el almacenamiento disponible en un grupo de volúmenes debemos crear dispositivos de bloques llamados *volúmenes lógicos* (*logical volumes* o LVs). Una vez creado, un volumen lógico es como cualquier otro dispositivo de bloques y, por tanto, se puede formatear para crear en él un sistema de ficheros de cualquier tipo e, incluso, podría formar parte de otro grupo de volúmenes, creando así infinitas posibilidades. Un volumen lógico también se divide en bloques que, en este caso, se llaman *extensiones lógicas* (*logical extents* o LEs), todas del mismo tamaño, el cual coincide con el de las extensiones físicas (4 MiB por omisión, como hemos dicho).

Cuando se crea un volumen lógico, sus extensiones lógicas se hacen corresponder con extensiones físicas libres. Normalmente, a cada extensión lógica le corresponde una (y solo una) extensión física, de tal manera que cualquier operación de lectura/escritura que se haga con una extensión lógica se terminará realizando realmente sobre su correspondiente extensión física. Pero también es posible hacer que a una extensión lógica le corresponda más de una física, que la correspondencia se haga según ciertas restricciones, etc. Todo esto da lugar a distintos tipos de volúmenes lógicos con características diferentes, como vamos a ver a continuación. La figura 3 muestra la estructura descrita hasta ahora de un sistema LVM, donde se pueden ver los elementos fundamentales: volúmenes físicos (PVs), extensiones físicas (PEs), extensiones lógicas (LEs), grupos de volúmenes (VGs) y volúmenes lógicos (LVs).

[2]Generalmente, para añadir un dispositivo de bloques a un grupo de volúmenes, no es necesario convertirlo previamente en un volumen físico, ya que, al añadirlo, se convertirá automáticamente. Por lo tanto, podríamos haber omitido la orden `pvcreate`.

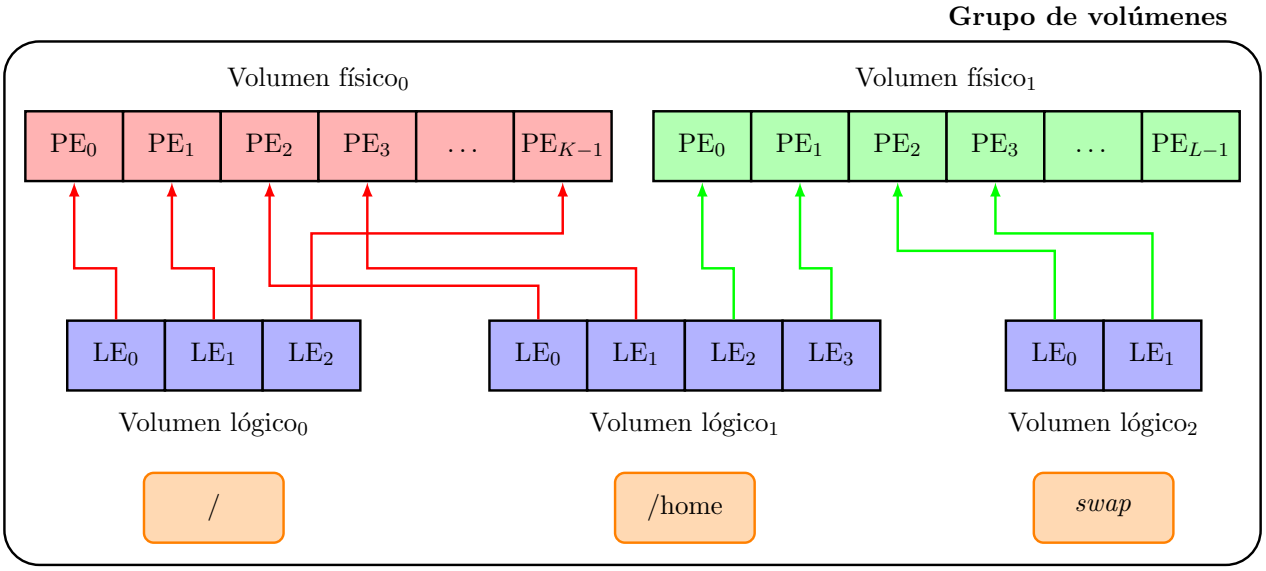

Figura 3: Arquitectura de un sistema LVM. El grupo de volúmenes tiene definidos tres volúmenes lógicos cuyas extensiones lógicas se corresponden con extensiones físicas repartidas entre los dos volúmenes físicos disponibles.

7.2. Volúmenes lógicos

Como acabamos de decir, existen distintos tipos de volúmenes lógicos. Aquí nos vamos a centrar en cuatro tipos[3] que vamos a llamar *lineal*, *repartido* (*striped*), *reflejado* o *en espejo* (*mirror*) y *raid5*.

7.2.1. Volumen lógico lineal

En un *volumen lógico lineal*, a cada extensión lógica le corresponde una única extensión física. No hay control en cómo se seleccionan las extensiones físicas: pueden estar todas en el mismo volumen físico, en volúmenes físicos distintos (del mismo disco o no), etc. La figura 4 muestra este tipo de volúmenes lógicos. Por ejemplo, podemos crear un volumen lógico lineal con un tamaño de 100 MiB y nombre `lv-lineal` ejecutando la siguiente orden:

```
# lvcreate -L 100M -n lv-lineal vg-iso
```

donde, como podemos ver, con `-L 100M` indicamos el tamaño y con `-n lv-linea` el nombre.

Para consultar las características del nuevo volumen lógico, como su tamaño, se usa `lvdisplay`. En este caso, vamos a añadir la opción `-m|--maps` para ver también el «mapa» que nos muestra cómo se corresponden las extensiones lógicas con físicas:

```
# lvdisplay --maps /dev/vg-iso/lv-lineal
  --- Logical volume ---
  LV Path                /dev/vg-iso/lv-lineal
  LV Name                lv-lineal
  VG Name                vg-iso
  ....
  LV Size                100,00 MiB
  Current LE             25
  Segments               1
  ....
  --- Segments ---
  Logical extents 0 to 24:
```

[3]En realidad, Linux soporta muchos tipos de volúmenes. Consulta la página de manual de `lvcreate` para ver todos los que hay disponibles.

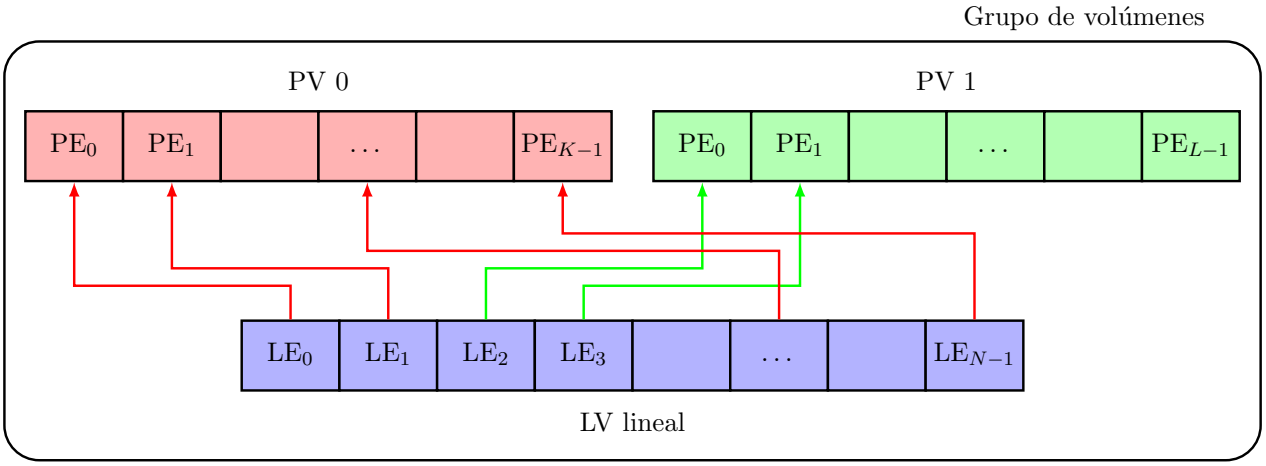

Figura 4: Un volumen lógico lineal. A cada extensión lógica le corresponde una única extensión física. Las extensiones lógicas se corresponden con extensiones en dos volúmenes físicos sin seguir ningún orden concreto.

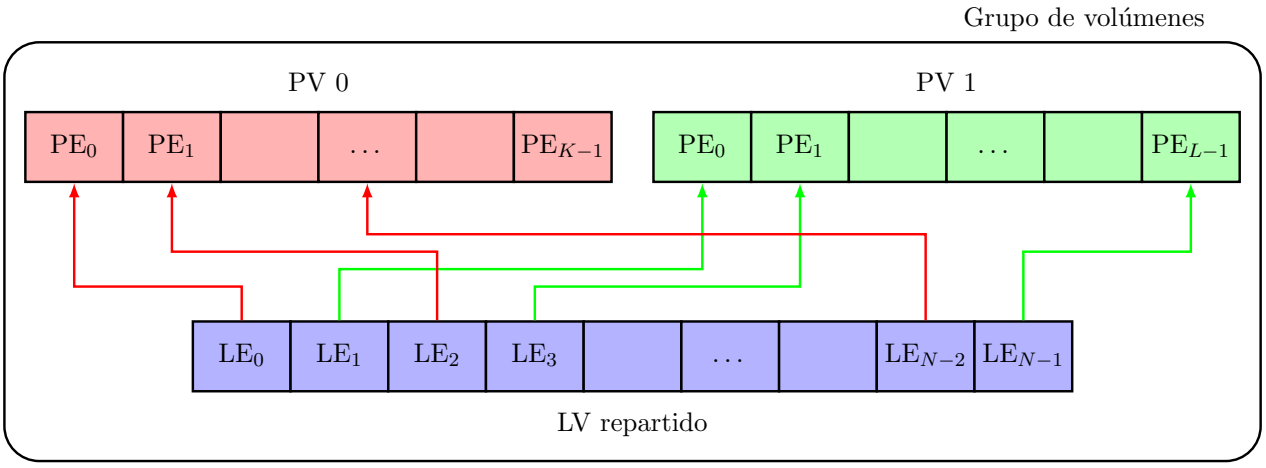

Figura 5: Un volumen lógico repartido (*striped*). A cada extensión lógica le corresponde una única extensión física. Como se ve, las extensiones lógicas se corresponden con extensiones en dos volúmenes físicos según un cierto orden. En concreto, LE_0 se asocia con PE_0 de PV 0, LE_1 con PE_0 de PV 1, LE_2 con PE_1 de PV 0, LE_3 con PE_1 de PV 1, y así sucesivamente.

```
Type                linear
Physical volume     /dev/sdb1
Physical extents    0 to 24
```

Observa la ruta que da nombre al nuevo dispositivo de bloques. Los volúmenes lógicos que creemos en el grupo `vg-iso` se representarán mediante ficheros en `/dev/vg-iso`. Estos ficheros serán enlaces simbólicos a ficheros especiales de bloques en `/dev`. Al final de la salida de la orden vemos que las 25 extensiones lógicas (de la 0 a la 24) de este volumen lógico se corresponden con las primeras 25 extensiones físicas del volumen físico `/dev/sdb1`.

7.2.2. Volumen lógico repartido

Un *volumen lógico repartido* se construye asignando también a cada extensión lógica una única extensión física (como en un volumen lógico lineal), con la peculiaridad de que las extensiones físicas de dos extensiones lógicas consecutivas se almacenan en volúmenes físicos de discos diferentes (ver figura 5). Esto hace que el rendimiento de un volumen lógico de este tipo sea mayor que el de un único disco, ya que las lecturas/escrituras se realizan usando varios discos a la vez, pero también hace que su fiabilidad sea menor, ya que, si se pierde un volumen físico (por un fallo de disco, por ejemplo), parte

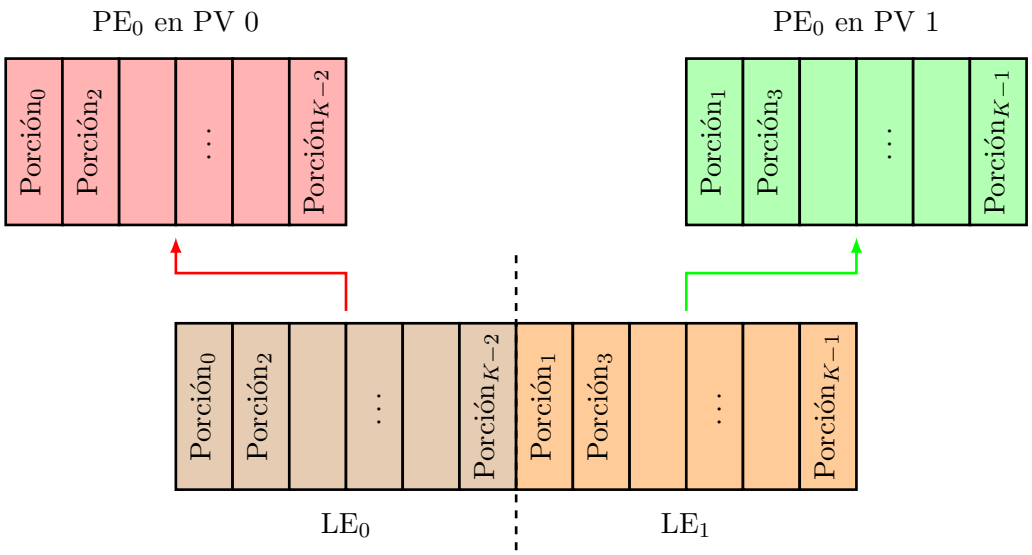

Figura 6: Funcionamiento de las porciones o *stripes* en un volumen lógico repartido que usa dos volúmenes físicos. Observa que, al cambiar de porción, se cambia de extensión lógica y, por tanto, de extensión física.

de la información almacenada en el volumen lógico se perderá también. De hecho, la probabilidad de que falle un volumen lógico repartido es mayor que la de un disco físico individual, ya que dicha probabilidad es aproximadamente la suma de las probabilidades de fallo de los discos individuales[4].

A la hora de crear un volumen lógico repartido, hay que usar la opción `-i|--stripes` para indicar cuántos volúmenes físicos se van a usar:

```
# lvcreate -i 2 -L 304M -n lv-repartido vg-iso
```

Aquí, por ejemplo, estamos creando un volumen lógico repartido, de nombre `lv-repartido`, con una capacidad de 304 MiB que usa extensiones físicas de dos volúmenes físicos (`-i 2`), lo que supone que cada volumen físico tendrá 152 MiB del volumen lógico.

Las extensiones de un volumen lógico repartido se dividen todavía en bloques más pequeños llamados *porciones*. El *tamaño de porción* (o *stripe size*) indica cada cuántos bytes consecutivos tenemos que cambiar de extensión lógica y, por tanto, de extensión física. Por omisión, el tamaño de porción es de 64 KiB. Este valor lo podemos cambiar al crear un volumen lógico repartido con la opción `-I`.

Para entender la necesidad de las porciones, vayamos de nuevo a la figura 5. Viendo esta figura, podemos entender que los primeros 4 MiB del volumen lógico repartido, almacenamos en su extensión lógica 0, se encuentran en la extensión física 0 del volumen físico 0, mientras que los siguiente 4 MiB, almacenados en su extensión lógica 1, se encuentran en la extensión física 0 del volumen físico 1, y así sucesivamente. Si esto fuera así, solo las lecturas o escrituras de más de 4 MiB consecutivos usarían con toda seguridad más de un volumen físico a la vez y, por tanto, obtendrían un rendimiento mayor que el de una operación más pequeña que usara un único volumen físico. Sin embargo, si queremos mejorar también el rendimiento de lecturas y escrituras pequeñas, serán necesario cambiar de extensión lógica y, en consecuencia, de extensión física no cada 4 MiB, sino en secuencias de bytes (porciones) más pequeñas.

La figura 6 muestra el funcionamiento de las porciones en el volumen lógico repartido de la figura 5, que usa dos volúmenes físicos. Como vemos, cada extensión lógica se divide en porciones, estando la porción 0 en la extensión lógica 0, la porción 1 en la extensión lógica 1, la porción 2 en la extensión lógica 0, y así sucesivamente. Al estar cada extensión lógica en una extensión física distinta, al cambiar de porción se cambia de extensión lógica y, por tanto, de extensión física. El resultado es que cualquier lectura o escritura con un tamaño de dos o más porciones usará los dos volúmenes físicos a la vez.

[4]La probabilidad de que alguno de entre n discos falle, con una probabilidad p de fallo de un disco individual, es realmente $1-(1-p)^n$, es decir, uno menos la probabilidad de que ningún disco falle.

Observa que, si al crear el volumen lógico repartido, hubiéramos puesto, por ejemplo, `300M` en lugar de `304M` como tamaño, el volumen lógico final habría tenido una capacidad igualmente de 304 MiB. Estos redondeos hacia arriba al crear volúmenes lógicos son necesarios a veces por diversos motivos que dependen del tipo de volumen lógico a crear. En nuestro caso, 300 MiB son 75 extensiones lógicas, por lo que, teniendo en cuenta lo que acabamos de ver de cómo se reparten las porciones, nuestro volumen lógico repartido usaría 37,5 extensiones de un volumen físico y otras 37,5 del otro volumen físico. Como eso supondría dejar sin usar media extensión en cada volumen físico, se opta por redondear hacia arriba hasta 304 MiB, ya que eso son 76 extensiones lógicas que se corresponden con exactamente 38 extensiones físicas en cada volumen físico.

7.2.3. Volumen lógico reflejado

En un *volumen lógico reflejado*, a cada extensión lógica le corresponden dos o más extensiones físicas, todas ellas con la misma información. Además, estas extensiones físicas se almacenan en volúmenes físicos distintos pertenecientes a discos distintos. De esta manera se consigue tolerancia a fallos, ya que, si un volumen físico falla y sus extensiones físicas se pierden, la información de cada extensión lógica todavía estará disponible en el resto de extensiones físicas. Aún así hay que tener en cuenta que se puede perder información debido a otras causas, como un borrado accidental de ficheros, por lo que es evidente que el uso de este tipo de volúmenes lógicos no elimina la necesidad de una buena política de copias de seguridad.

En cuanto al rendimiento, en un volumen lógico reflejado las escrituras tardan lo mismo que en un disco físico individual, pues se tiene que hacer una copia en todos los discos (estas copias se hacen en paralelo). En cambio, se mejora el rendimiento de las lecturas pues la lectura de un extensión lógica puede ir a un volumen físico en un disco y la de otra al otro volumen físico en el otro disco.

Para crear un volumen lógico reflejado hay que usar la opción `-m|--mirrors` de `lvcreate`. Así, podemos crear un volumen lógico de este tipo de nombre `lv-reflejado`, con un tamaño de 200 MiBs y dos copias por extensión lógica (es decir, usando dos volúmenes físicos en discos distintos), ejecutando:

```
# lvcreate -m 1 -L 200M -n lv-reflejado vg-iso
```

Observa que en la opción `-m` hemos puesto el *número de copias adicionales (o de volúmenes físicos)* a utilizar, y no el total de volúmenes físicos. Si consultamos las características de este volumen lógico:

```
# lvdisplay --maps /dev/vg-iso/lv-reflejado
  --- Logical volume ---
  LV Path                /dev/vg-iso/lv-reflejado
  LV Name                lv-reflejado
  VG Name                vg-iso
  ....
  LV Size                200,00 MiB
  Current LE             50
  Mirrored volumes       2
  Segments               1
  ....
  --- Segments ---
  Logical extents 0 to 49:
    Type                raid1
    Monitoring          monitored
    Raid Data LV 0
      Logical volume    lv-reflejado_rimage_0
      Logical extents   0 to 49
    Raid Data LV 1
      Logical volume    lv-reflejado_rimage_1
      Logical extents   0 to 49
```

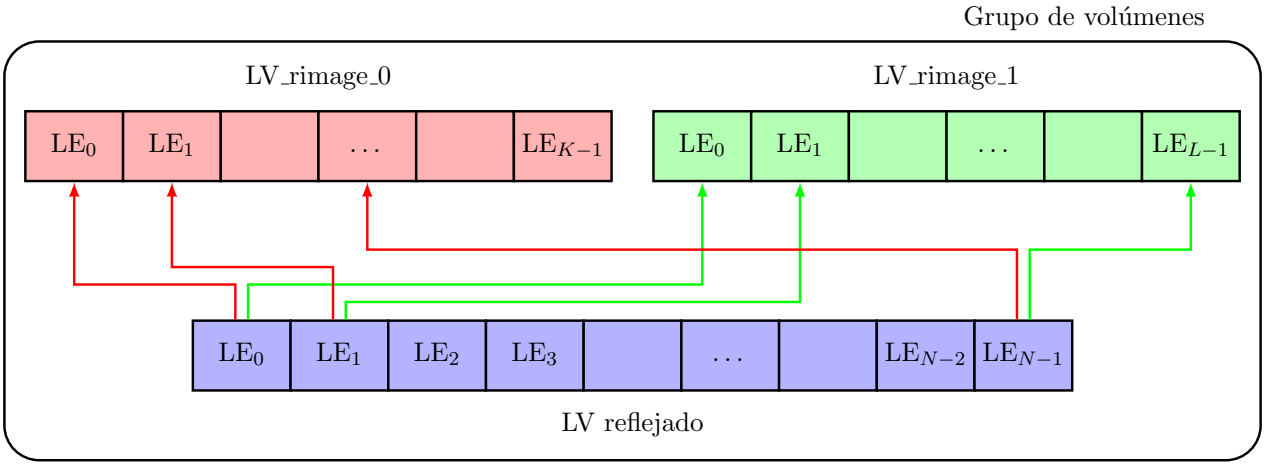

Figura 7: Un volumen lógico reflejado (*mirror*). A cada extensión lógica le corresponden dos extensiones lógicas, cada una en un volumen lógico interno distinto. Estos volúmenes internos son, a su vez, volúmenes lógicos lineales almacenados en volúmenes físicos que no comparten disco, por lo que, al final, a cada extensión lógica le corresponden dos extensiones físicas en discos distintos.

```
  Raid Metadata LV 0  lv-reflejado_rmeta_0
  Raid Metadata LV 1  lv-reflejado_rmeta_1
```

veremos que tiene un tamaño de 50 extensiones lógicas, que ocupan 50 extensiones `lógicas` (de 0 a 49) en el volumen lógico `lv-reflejado_rimage_0` y otras 50 (también de 0 a 49) en el volumen lógico `lv-reflejado_rimage_1`. Es decir, para implementar este tipo de volúmenes lógicos, el propio LVM de Linux usa volúmenes lógicos internos (ver figura 7), que no son sino volúmenes lógicos lineales. Podemos ver también que hay otros dos volúmenes lógicos, `lv-reflejado_rmeta_0` y `lv-reflejado_rmeta_1`, cada uno con una extensión lógica de tamaño, que LVM utiliza para los metadatos necesarios para el control de este tipo de volúmenes. Por lo tanto, el volumen lógico `/dev/vg-iso/lv-reflejado` de 50 extensiones lógicas (200 MiB) de tamaño termina consumiendo 102 extensiones físicas (408 MiB) en el grupo de volúmenes. Si usamos la opción `-m|--maps` de `pvdisplay`, podremos ver cómo se distribuyen esos volúmenes lógicos internos entre los volúmenes físicos:

```
# pvdisplay --maps
  --- Physical volume ---
  PV Name               /dev/sdb1
  ....
  --- Physical Segments ---
  Physical extent 0 to 24:
    Logical volume      /dev/vg-iso/lv-lineal
    Logical extents     0 to 24
  Physical extent 25 to 62:
    Logical volume      /dev/vg-iso/lv-repartido
    Logical extents     0 to 75
  Physical extent 63 to 63:
    Logical volume      /dev/vg-iso/lv-reflejado_rmeta_0
    Logical extents     0 to 0
  Physical extent 64 to 113:
    Logical volume      /dev/vg-iso/lv-reflejado_rimage_0
    Logical extents     0 to 49
  Physical extent 114 to 254:
    FREE
```

```
  --- Physical volume ---
  PV Name               /dev/sdc1
  ....
  --- Physical Segments ---
  Physical extent 0 to 37:
    Logical volume      /dev/vg-iso/lv-repartido
    Logical extents     0 to 75
  Physical extent 38 to 38:
    Logical volume      /dev/vg-iso/lv-reflejado_rmeta_1
    Logical extents     0 to 0
  Physical extent 39 to 88:
    Logical volume      /dev/vg-iso/lv-reflejado_rimage_1
    Logical extents     0 to 49
  Physical extent 89 to 254:
    FREE
  ....
```

Aunque lo habitual es que un volumen lógico reflejado use solo dos volúmenes físicos en discos diferentes, nada impide usar más discos. Simplemente, en lugar de haber dos copias de cada extensión lógica habrá varias. Se mejora así la fiabilidad y el rendimiento de las lecturas, pero no se consigue aumentar ni la capacidad del volumen lógico ni el rendimiento de las escrituras.

7.2.4. Volumen lógico raid5

Finalmente, un *volumen lógico raid5* funciona como un volumen lógico repartido al que se le añade paridad, de tal manera que, aunque un volumen físico deje de funcionar, el volumen lógico seguirá funcionando. Este tipo de volúmenes necesita, al menos, tres volúmenes físicos en dispositivos distintos. Podemos crear uno ejecutando lo siguiente:

```
# lvcreate --type raid5 -i 2 -L 240M -n lv-raid5 vg-iso
```

Como podemos ver, aunque indiquemos `-i 2`, no se usarán dos volúmenes físicos en dispositivos distintos, sino 3. En general, si se indica `-i` N, se usarán $N+1$ volúmenes físicos independientes. De nuevo, si consultamos las características de este volumen lógico:

```
# lvdisplay --maps /dev/vg-iso/lv-raid5
  --- Logical volume ---
  LV Path                /dev/vg-iso/lv-raid5
  ....
  Segments               1
  ....
  --- Segments ---
  Logical extents 0 to 59:
    Type                raid5
    Monitoring          monitored
    Raid Data LV 0
      Logical volume    lv-raid5_rimage_0
      Logical extents   0 to 29
    Raid Data LV 1
      Logical volume    lv-raid5_rimage_1
      Logical extents   0 to 29
    Raid Data LV 2
      Logical volume    lv-raid5_rimage_2
      Logical extents   0 to 29
    Raid Metadata LV 0  lv-raid5_rmeta_0
```

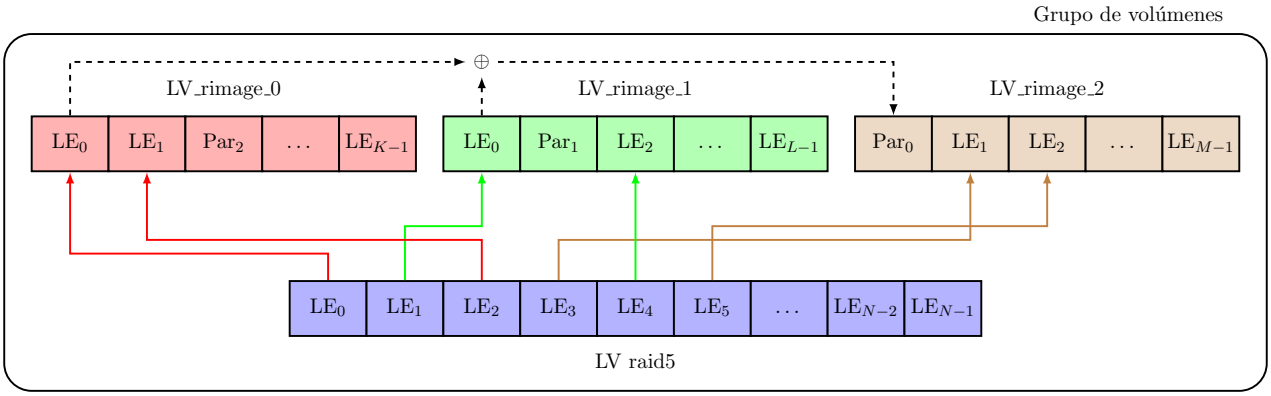

Figura 8: Un volumen lógico raid5. A cada extensión lógica le corresponde una única extensión lógica en un volumen lógico interno siguiendo el orden mostrado. Como se ve, cada dos extensiones lógicas se calcula la paridad de ambas, que se almacena en el volumen lógico interno no usado por esas dos extensiones lógicas. Como en un volumen lógico reflejado, los volúmenes lógicos internos son de tipo lineal que no comparten discos físicos.

```
Raid Metadata LV 1  lv-raid5_rmeta_1
Raid Metadata LV 2  lv-raid5_rmeta_2
```

veremos que tiene un tamaño de 60 extensiones lógicas (que equivalen a esos 240 MiB) y que internamente se implementa con 6 volúmenes lógicos: 3 para guardar datos y paridad (ver figura 8) y otros 3 para guardar los metadatos necesarios para este tipo de volúmenes lógicos. Si comparamos la salida de `vgdisplay` antes y después de crear este volumen, veremos que se han consumido 93 extensiones físicas, que se corresponden con: 60 para guardar datos, 30 para guardar paridad y una para cada uno de los volúmenes lógicos que almacenan metadatos (3 en total). De nuevo, podemos ejecutar `pvdisplay -m` para ver cómo se distribuyen esos volúmenes lógicos internos entre los volúmenes físicos.

Respecto a usar un único disco, un volumen lógico raid5 ofrece, por tanto, tolerancia a fallos (puede fallar un volumen físico sin que se pierda información, como en un volumen lógico reflejado), mayor capacidad (usando N volúmenes físicos, la capacidad de este tipo de volúmenes lógicos es la de $N-1$ volúmenes físicos) y mejor rendimiento (hay varios discos que nos permiten hacer lecturas/escrituras en paralelo). Al igual que en un volumen lógico repartido, en uno raid5 las extensiones lógicas se dividen en porciones para mejorar el rendimiento de las lecturas y escrituras de menos de 4 MiB. El tamaño de porción por omisión también es 64 KiB y se puede cambiar al crear el volumen lógico con la opción `-I`, como ya se ha indicado.

7.3. Cambio de tamaño de un sistema LVM

Una característica importante de los volúmenes lógicos es que se pueden agrandar, concatenándoles nuevas extensiones lógicas, o encoger, quitándoles extensiones lógicas cuyas extensiones físicas se liberan y se devuelven al repositorio de extensiones libres dentro del grupo de volúmenes. Al agrandar un volumen lógico, las nuevas extensiones lógicas pueden estar asociadas a extensiones físicas que no tienen por qué ser contiguas ni tienen por qué encontrarse en los mismos volúmenes físicos que las ya utilizadas. Esto permite a los volúmenes lógicos crecer sin tener que mover las extensiones lógicas que ya poseen. Generalmente, es posible cambiar el tamaño de un volumen lógico mientras está en uso.

El aumento de tamaño de un volumen lógico se realiza con la orden `lvextend`. Por ejemplo, podemos añadir 200 MiB más a nuestro volumen `lv-reflejado` ejecutando la orden:

```
# lvextend -L +200M /dev/vg-iso/lv-reflejado
```

Observa que el incremento se ha indicado usando un signo «+». Si suprimimos este carácter, el nuevo tamaño del volumen lógico sería exactamente 200 MiB y no 200 MiB más. En nuestro ejemplo, esto supondría no cambiar el tamaño pues ya tiene 200 MiB. En lugar de `-L`, también podemos usar `-l`

para indicar un tamaño en extensiones lógicas. Así, con esta opción, la siguiente orden sería equivalente a la anterior (recordemos que el tamaño de las extensiones es 4 MiB):

```
# lvextend -l +50 /dev/vg-iso/lv-reflejado
```

La opción `-l` también permite indicar un incremento de tamaño en función del espacio libre en el grupo de volúmenes, del tamaño total del grupo de volúmenes, etc. Por ejemplo, la siguiente orden incrementa el tamaño del volumen `lv-raid5` en un 10 % del tamaño del grupo de volúmenes:

```
# lvextend -l +10%VG /dev/vg-iso/lv-raid5
```

mientras que la siguiente lo hace en un 10 % del espacio libre en ese momento:

```
# lvextend -l +10%FREE /dev/vg-iso/lv-raid5
```

Hay que tener en cuenta que una cosa es el tamaño de un volumen lógico y otra el tamaño del sistema de ficheros que pueda contener. Así, si incrementamos el tamaño de un volumen lógico que contiene un sistema de ficheros, y no redimensionamos este, el sistema de ficheros seguirá teniendo el mismo tamaño y, por tanto, no aprovechará todo el espacio disponible en el volumen lógico. Hablaremos de los sistemas de ficheros y de cómo redimensionarlos en el siguiente boletín, donde también veremos que la orden `lvextend` posee la opción `-r|--resize` que permite redimensionar el sistema de ficheros contenido en un volumen lógico a la misma vez que se aumenta el tamaño de dicho volumen.

Para reducir el tamaño de un volumen lógico, debemos usar la orden `lvreduce`. Esta orden elimina extensiones lógicas del volumen lógico indicado. Así, podemos quitarle 10 MiB a nuestro volumen `lv-repartido` ejecutando:

```
# lvreduce -L -10M /dev/vg-iso/lv-repartido
```

Las opciones vistas en `lvextend` para aumentar el tamaño de un volumen lógico también las podemos usar con `lvreduce`. Además, de forma similar a lo que hemos explicado antes, si el volumen lógico contiene un sistema de ficheros, habrá que reducir el tamaño de este antes de reducir el del volumen lógico. También describiremos cómo hacer esto en el siguiente boletín.

Es posible también aumentar el tamaño de un grupo de volúmenes, añadiendo más volúmenes físicos con la orden `vgextend`, o reducir su tamaño, eliminando volúmenes físicos con la orden `vgreduce`. En este segundo caso, si se están usando extensiones físicas del volumen a eliminar, será necesario mover su contenido a otras extensiones físicas libres de otros volúmenes físicos usando `pvmove`. Habitualmente, el gestor de volúmenes lógicos puede hacer este movimiento en vivo. Si, además, el hardware permite cambiar los discos en caliente, entonces es posible actualizar o reemplazar dispositivos de almacenamiento sin parar el sistema.

7.4. Eliminación de elementos en un sistema LVM

Finalmente, podemos eliminar volúmenes lógicos con la orden `lvremove` y grupos de volúmenes enteros con `vgremove`. Así, si ejecutamos:

```
# lvremove /dev/vg-iso/lv-raid5
```

eliminaremos nuestro volumen lógico `lv-raid5` (siempre que no se esté usando), mientras que si ejecutamos:

```
# vgremove vg-iso
```

eliminaremos no solo nuestro grupo de volúmenes sino, por supuesto, también todos los volúmenes lógicos que todavía quedan en él.

En el caso de los volúmenes físicos, debemos usar la orden `pvremove` para eliminarlos del sistema LVM. Tras esta orden, el dispositivo de bloques seguirá existiendo, aunque ya no se tendrá en cuenta como posible volumen físico.

8. Dispositivos de bloques *loop*

Si los ficheros especiales de bloques permiten acceder a los dispositivos de bloques como si de ficheros regulares se tratara, los dispositivos de bloques *loop* permiten lo contrario, es decir, que un fichero regular sea visto como un dispositivo de bloques y, por tanto, se pueda formatear con un sistema de ficheros, montar, añadir a un grupo de volúmenes, etc. Esto permite experimentar con dispositivos de bloques aun cuando no tenemos discos o particiones disponibles en nuestro ordenador.

Por ejemplo, imaginemos que queremos trabajar con particiones sin usar ninguno de nuestros discos. Para ello, podemos crear un fichero de, digamos, 200 MiB, crear un dispositivo loop asociado y trabajar con el nuevo dispositivo de bloques. Lo primero que hacemos es crear el fichero con la orden `dd` de la siguiente manera:

```
# dd if=/dev/zero of=/root/disco200M.img bs=1M count=200
```

Con el parámetro `if=` indicamos el fichero de entrada, o fichero origen, del que queremos leer la información a copiar. En nuestro caso es el fichero especial `/dev/zero`, al que podemos llamar, la «fuente inagotable de ceros», ya que podemos leer continuamente de él y siempre nos devolverá ceros. Con el parámetro `of=` establecemos el fichero de salida o destino de la copia, que aquí es el fichero regular `/root/disco200M.img` (podemos usar el nombre que queramos). Con `bs=` fijamos el tamaño de los bloques a copiar del origen al destino (1 MiB en el ejemplo) y con `count=` cuántos de esos bloques queremos copiar. Así, 200 bloques de 1 MiB son 200 MiB. Podemos indicar otros valores para el tamaño de bloque y el número de bloques; lo importante es que la multiplicación de ambos valores dé el tamaño deseado.

Una vez creado el fichero que dará soporte a nuestro «disco», debemos asociarlo a un dispositivo loop, lo que hará que se cree también el fichero especial de bloques que lo represente. Esto lo hacemos ejecutando la siguiente orden:

```
# losetup -fP /root/disco200M.img
```

La opción `-f` hace que se utilice el primer dispositivo loop libre. La opción `-P` sirve para que, si se crean particiones, también se creen los ficheros especiales de bloques que las representan y así poder trabajar con ellas. En lugar de `-f`, podemos especificar un dispositivo loop concreto así:

```
# losetup -P /dev/loop0 /root/disco200M.img
```

Podemos ver los dispositivos loop actuales ejecutando:

```
# losetup -a
/dev/loop0: [64768]:260293 (/root/disco200M.img)
```

A partir de ahora, podemos usar `/dev/loop0` como cualquier otro disco. Como vemos, solo se está usando el primer dispositivo loop. El resto de dispositivos loop (desde `/dev/loop1` hasta `/dev/loop7`) están libres.

Finalmente, podemos eliminar el dispositivo loop (siempre que no se esté usando) con la orden:

```
# losetup -d /dev/loop0
```

o eliminarlos todos con:

```
# losetup -D
```

Como otras, la orden `losetup` tiene más opciones útiles que podemos ver en su página de manual.

9. Ejercicios

La máquina virtual de Fedora que empleamos en las prácticas trae configurados cuatro discos duros: uno de 10 GiB, en el que está instalado el sistema operativo, y tres de 1 GiB para pruebas. Estos últimos están sin particionar y los usaremos en algunos ejercicios.

9.1. Particiones y dispositivos loop

1. En este ejercicio vamos a crear varias particiones en un dispositivo loop que hay que crear previamente.

   1.1 Crea el fichero regular `/root/disco100M.img` de 100 MiB y asócialo al dispositivo `/dev/loop0`. Asegúrate de que ha quedado asociado. ¿Aparece el nuevo dispositivo en la salida de la orden `lsblk`?

   1.2 Particiona el nuevo dispositivo con las siguientes particiones usando particionamiento DOS (todas las particiones deben empezar en el primer sector libre disponible):

   - Partición 1: partición primaria de 10 MiB.
   - Partición 2: partición extendida que ocupe el resto del espacio del dispositivo.
   - Partición 5: partición lógica de 5 MiB y tipo «W95 FAT32 (LBA)».
   - Partición 6: partición lógica que ocupe el resto del espacio de la partición extendida y tipo «Linux LVM».

   1.3 Desde la línea de órdenes, visualiza la tabla de particiones del dispositivo `/dev/loop0` (y solo de ese dispositivo) con `fdisk`. Lista también las particiones de `/dev/loop0` (y solo de `/dev/loop0`) con `lsblk`. ¿Ves alguna diferencia en los tamaños mostrados en cada caso?

   1.4 Lista, usando `ls -l`, los nuevos ficheros especiales de bloques que se han creado en `/dev`. ¿Aparecen ficheros especiales para las particiones? (Debería ser así; si no aparecen, piensa dónde puede estar el error.)

   1.5 Repite el ejercicio anterior, pero usando ahora `lsblk`.

   1.6 Ejecuta ahora `lsblk -f`. ¿Aparece alguna información bajo la columna UUID para alguna de las particiones del dispositivo *loop*? Razona tu respuesta.

   1.7 Reinicia el ordenador (sin crear una máquina virtual nueva) y, sin hacer nada más, lista los dispositivos de bloques disponibles. Deberías ver que el dispositivo loop creado y sus particiones no aparecen, aunque el fichero `/root/disco100M.img` sigue ahí. Asocia de nuevo el fichero a un dispositivo loop (deja que el sistema le asigne automáticamente el primero disponible) y lista de nuevo los dispositivos de bloques disponibles. ¿Aparecen ahora todas las particiones creadas anteriormente?

   1.8 Finalmente, desliga el dispositivo loop del fichero y comprueba que la operación se ha realizado con éxito. Borra después el fichero creado.

2. Ahora vamos a hacer un ejercicio similar al anterior pero utilizando el disco `/dev/sdc` de la máquina virtual.

   2.1 Utilizando particionamiento GPT, crea en el disco `/dev/sdc` las siguiente particiones (todas las particiones deben empezar en el primer sector libre disponible):

   - Partición 1 de 500 MiB.
   - Partición 2 de 300 MiB y tipo «Linux swap».
   - Partición 3 de 200 MiB y tipo «Microsoft basic data».

   2.2 Desde la línea de órdenes, visualiza la tabla de particiones de `/dev/sdc` (y solo de ese dispositivo) con `fdisk`. Lista también las particiones de `/dev/sdc` (y solo de `/dev/sdc`) con `lsblk`. ¿Ves ahora alguna diferencia en los tamaños mostrados en cada caso?

   2.3 Lista, usando `ls -l`, los nuevos ficheros especiales de bloques que se han creado en `/dev`.

   2.4 Repite el ejercicio anterior, pero usando ahora `lsblk`.

   2.5 Finalmente, borra todas las particiones que has creado en `/dev/sdc`. Después, desde la línea de órdenes, comprueba con `fdisk` que ya no hay particiones en ese dispositivo.

9.2. Sistemas LVM

Para realizar los siguientes ejercicios, usa GPT y crea dos particiones en `/dev/sdb`: una de 500 MiB y otra que ocupe el resto del espacio libre. El tipo de ambas debe ser «Linux LVM». Haz lo mismo con `/dev/sdc` y `/dev/sdd`.

3. En primer lugar, comprueba qué grupo de volúmenes y qué volúmenes físicos y lógicos existen ya en el sistema. Indica de qué tipo son los volúmenes lógicos y cómo se corresponden sus extensiones lógicas con las de los volúmenes físicos.

4. Crea un grupo de volúmenes llamado `vg-ejercicios` formado por las primeras particiones de los tres discos de pruebas, es decir, `/dev/sdb1`, `/dev/sdc1` y `/dev/sdd1`.

5. Crea en `vg-ejercicios` un volumen lógico repartido de 300 MiB, llamado `lv-repartido-3d`, que use los tres volúmenes físicos. Indica cómo se distribuyen las extensiones lógicas entre los volúmenes físicos y compara el tamaño total del volumen lógico con lo ocupado en cada volumen físico.

6. Crea ahora en `vg-ejercicios` un volumen lógico reflejado de 100 MiB, llamado `lv-reflejado-2d`, que use solo dos volúmenes físicos. Indica qué volúmenes físicos se han usado para almacenar las extensiones lógicas y qué volúmenes lógicos internos se almacenan en cada volumen físico.

7. Añade ahora al grupo de volúmenes `vg-ejercicios` el dispositivo `/dev/sdb2`. Comprueba que el grupo tiene ahora cuatro volúmenes.

8. Intenta crear en `vg-ejercicios` un volumen lógico raid5 de 50 MiB, llamado `lv-raid5-4d`, que use los cuatro volúmenes físicos que hay. ¿Puedes? ¿Debería ser así? ¿Por qué? A la hora de responder, analiza si el raid5 sobreviviría a la pérdida por completo de cualquiera de los discos que contienen las particiones que integran el grupo de volúmenes (recuerda que estos discos son `/dev/sdb`, `/dev/sdc` y `/dev/sdd`).

9. Comprueba cuántas extensiones físicas quedan en `vg-ejercicios`. Después, intenta crear en ese grupo de volúmenes un volumen lógico raid5, llamado `lv-raid5-3d`, que ocupe todo el espacio disponible en el grupo de volúmenes y use tres volúmenes físicos. ¿Puedes? Si se llega a crear, ¿se crea con el tamaño solicitado? ¿Por qué?

10. Añade también al grupo de volúmenes `vg-ejercicios` el dispositivo `/dev/sdc2` e incrementa el tamaño de `lv-reflejado-2d` en 100 MiB. ¿Se usan los mismos volúmenes lógicos internos que antes o se han añadido nuevos? ¿Qué tamaño tienen los volúmenes lógicos internos (ya sea existentes o nuevos)? ¿Qué volúmenes físicos usa ahora el volumen lógico? ¿Se sigue garantizando que, si un disco falla, el volumen lógico va a seguir funcionando?

11. Añade a `vg-ejercicios` el dispositivo `/dev/sdd2` y haz lo necesario para eliminar `/dev/sdb2` por completo de dicho grupo de volúmenes, de tal manera que deje incluso de ser considerado como un potencial volumen físico.

12. Añade ahora `/dev/sdb2` al grupo de volúmenes `fedora` que ya existía en el sistema. Después, consulta el tamaño del sistema de ficheros que hay en el volumen lógico `root` de dicho grupo de volúmenes ejecutando `df /` (mira el dato bajo la columna `bloques de 1K`). A continuación, aumenta el tamaño de ese volumen lógico para que tanto el volumen lógico como el sistema de ficheros que contiene usen todo el nuevo espacio disponible. Seguidamente, comprueba el nuevo tamaño del volumen lógico y también si dicho volumen usa el volumen físico añadido. Finalmente, ejecuta otra vez `df /` y mira si el tamaño del sistema de ficheros es ahora mayor.