Empilement de périphériques en mode bloc

La démo est modifiée pour vous permettre de la rejouer sur votre machine en tant que `root`, sans utiliser de disque physique externe, clef USB, carte sd ou autre, uniquement des périphérique boucle.

Si vous utilisez WSL sous windows, il faut passer à WSL2 (ou mieux : profitez de l'occasion pour passer à `virtualbox`).

On part de deux périphériques `loop` qu'on va découper, coller, chiffrer, etc

- Point de départ : 2 disques de `100 MiB` chacun
- Point d'arrivée :
  - un périphérique en mode bloc chiffré de `70 MiB`, formaté `ext4`
  - un périphérique en mode bloc de `70 MiB`, formaté `xfs`
  - un périphérique en mode bloc de `50 MiB`, formaté `vfat`

On se place dans le répertoire temporaire `/tmp` :

```
# cd /tmp
```

On crée un fichier `/tmp/disque1` rempli de zéros de taille `100 MiB`. Pour cela, on utilise le fichier spécial `/dev/zero` qui est un périphérique caractère qui génère une infinité de zéros. On utilise la commande `dd` car il faut bien s'arrêter à un moment.

```
# dd if=/dev/zero of=disque1 bs=1MiB count=100
100+0 enregistrements lus
100+0 enregistrements écrits
104857600 octets (105 MB, 100 MiB) copiés, 0,0612908 s, 1,7 GB/s
```

Remarque : dans la démo originale, on avait d'abord testé :

```
# cp /dev/zero disque1
```

et on s'était rendu compte que le système de fichiers sur lequel était monté `/tmp` s'était rempli très vite, avec un fichier `disque1` qui a pris toute la place avant de créer une erreur. Ne le faites pas sur les machines de TP (ou alors allez dans `/dev/shm` et n'oubliez pas d'effacer le fichier ensuite).

On peut voir que le fichier est plein de zéros :

```
# hexdump disque1
0000000 0000 0000 0000 0000 0000 0000 0000 0000
*
6400000
```

On fait pareil avec le fichier `/tmp/disque2`, ou alors on recopie le contenu de `disque1` dans `disque2` :

```
# cp disque1 disque2
```

Ces fichiers sont des fichiers réguliers, on peut les promouvoir en périphériques en mode bloc :

```
# losetup -f disque1
```

On regarde quel périphérique boucle est associé à `disque1` :

```
# losetup --associated disque1
/dev/loop0: [0031]:3565662 (/tmp/disque1)
```

On fait pareil pour `disque2` :

```
# losetup -f disque2
```

On liste les périphériques boucles existants :

```
# losetup --list
NAME       SIZELIMIT OFFSET AUTOCLEAR RO BACK-FILE    DIO LOG-SEC
/dev/loop1         0      0         0  0 /tmp/disque2   0     512
/dev/loop0         0      0         0  0 /tmp/disque1   0     512
```

On peut voir que deux périphériques bloc sont apparus :

```
# lsblk
NAME           MAJ:MIN RM   SIZE RO TYPE  MOUNTPOINT
...
loop0            7:0    0   100M  0 loop
loop1            7:1    0   100M  0 loop
```

On crée une table de partitions de type `msdos` sur `/dev/loop0` (`man parted`) :

```
# parted /dev/loop0 mklabel msdos
Information: You may need to update /etc/fstab.
```

On peut voir qu'une *en-tête* (en: *header*) a été écrite au début du fichier :

```
# hexdump disque1
0000000 b8fa 1000 d08e 00bc b8b0 0000 d88e c08e
0000010 befb 7c00 00bf b906 0200 a4f3 21ea 0006
0000020 be00 07be 0438 0b75 c683 8110 fefe 7507
0000030 ebf3 b416 b002 bb01 7c00 80b2 748a 8b01
0000040 024c 13cd 00ea 007c eb00 00fe 0000 0000
0000050 0000 0000 0000 0000 0000 0000 0000 0000
*
00001b0 0000 0000 0000 0000 e571 5a09 0000 0000
00001c0 0000 0000 0000 0000 0000 0000 0000 0000
*
00001f0 0000 0000 0000 0000 0000 0000 0000 aa55
0000200 0000 0000 0000 0000 0000 0000 0000 0000
*
6400000
```

On crée deux partitions primaires de taille `50 MiB` :

```
# parted /dev/loop0 mkpart primary 0 50MiB
Warning: The resulting partition is not properly aligned for best performance.
Ignore/Cancel? C
```

Il est pas content, on annule (`C`) et on re-éssaye :

```
# parted /dev/loop0 mkpart primary 1MiB 50MiB
Information: You may need to update /etc/fstab.

# parted /dev/loop0 mkpart primary 50MiB 100%
Information: You may need to update /etc/fstab.
```

On liste (`"ls"`) les périphériques en mode bloc (`"blk"`) pour voir ou on en est :

```
# lsblk
NAME           MAJ:MIN RM   SIZE RO TYPE  MOUNTPOINT
...
loop0            7:0    0   100M  0 loop
├─loop0p1      259:0    0    49M  0 part
```

```
└─loop0p2      259:1    0    50M  0 part
loop1          7:1      0   100M  0 loop
```

On peut aussi remarquer que le fait d'avoir créé deux partitions, n'a fait que modifier la table de partitions, mais par exemple, rien n'est écrit au milieu du disque (à la frontière entre les partitions) :

```
# hexdump disque1
0000000 b8fa 1000 d08e 00bc b8b0 0000 d88e c08e
0000010 befb 7c00 00bf b906 0200 a4f3 21ea 0006
0000020 be00 07be 0438 0b75 c683 8110 fefe 7507
0000030 ebf3 b416 b002 bb01 7c00 80b2 748a 8b01
0000040 024c 13cd 00ea 007c eb00 00fe 0000 0000
0000050 0000 0000 0000 0000 0000 0000 0000 0000
*
00001b0 0000 0000 0000 0000 e571 5a09 0000 2000
00001c0 0021 5f83 0619 0800 0000 8800 0001 5f00
00001d0 061a be83 0c32 9000 0001 9000 0001 0000
00001e0 0000 0000 0000 0000 0000 0000 0000 0000
00001f0 0000 0000 0000 0000 0000 0000 0000 aa55
0000200 0000 0000 0000 0000 0000 0000 0000 0000
*
6400000
```

Bien sur, c'est le même contenu que `/dev/loop0` :

```
# hexdump /dev/loop0
0000000 b8fa 1000 d08e 00bc b8b0 0000 d88e c08e
0000010 befb 7c00 00bf b906 0200 a4f3 21ea 0006
0000020 be00 07be 0438 0b75 c683 8110 fefe 7507
0000030 ebf3 b416 b002 bb01 7c00 80b2 748a 8b01
0000040 024c 13cd 00ea 007c eb00 00fe 0000 0000
0000050 0000 0000 0000 0000 0000 0000 0000 0000
*
00001b0 0000 0000 0000 0000 e571 5a09 0000 2000
00001c0 0021 5f83 0619 0800 0000 8800 0001 5f00
00001d0 061a be83 0c32 9000 0001 9000 0001 0000
00001e0 0000 0000 0000 0000 0000 0000 0000 0000
00001f0 0000 0000 0000 0000 0000 0000 0000 aa55
0000200 0000 0000 0000 0000 0000 0000 0000 0000
*
6400000
```

On peut voir que l'espace occupé par les partitions n'a que des zéros :

```
# hexdump /dev/loop0p1
0000000 0000 0000 0000 0000 0000 0000 0000 0000
*
3100000
```

On va utiliser `LVM` pour recoller la première partition de `/dev/loop0` (`50 MiB`) et `/dev/loop1` (`100 MiB`), redécouper l'ensemble en deux périphériques de tailles `70 MiB` chacun (voir le schéma des commandes `LVM` dans les slides du cours) :

On transforme les deux périphériques en volumes physiques `LVM` :

```
# pvcreate /dev/loop0p1
  Physical volume "/dev/loop0p1" successfully created.
```

On peut observer que ça a écrit des choses au début de la partition (notez les indications) :

```
# hexdump -C /dev/loop0p1
```

```
00000000  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000200  4c 41 42 45 4c 4f 4e 45  01 00 00 00 00 00 00 00  |LABELONE........|
00000210  12 2c 0d 55 20 00 00 00  4c 56 4d 32 20 30 30 31  |.,.U ...LVM2 001|
00000220  33 4b 34 39 62 6a 37 63  32 30 75 31 45 65 76 39  |3K49bj7c20u1Eev9|
00000230  32 42 58 62 37 4c 46 39  4b 67 72 73 61 48 42 52  |2BXb7LF9KgrsaHBR|
00000240  00 00 10 03 00 00 00 00  00 00 10 00 00 00 00 00  |................|
00000250  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000260  00 00 00 00 00 00 00 00  00 10 00 00 00 00 00 00  |................|
00000270  00 f0 0f 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000280  00 00 00 00 00 00 00 00  02 00 00 00 00 00 00 00  |................|
00000290  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00001000  16 d6 8e db 20 4c 56 4d  32 20 78 5b 35 41 25 72  |.... LVM2 x[5A%r|
00001010  30 4e 2a 3e 01 00 00 00  00 10 00 00 00 00 00 00  |0N*>............|
00001020  00 f0 0f 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00001030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
03100000
```

On fait pareil pour `/dev/loop1` :

```
# pvcreate /dev/loop1
  Physical volume "/dev/loop1" successfully created.
```

On les fusionne en un volume group `LVM` nommé `vg_disque` :

```
# vgcreate vg_disque /dev/loop0p1 /dev/loop1
  Volume group "vg_disque" successfully created
```

Si vous refaites un `hexdump -C` sur les deux périphériques bloc, vous verrez les indications sur le volume group (exercice).

On découpe le volume groupe `LVM` en deux volumes logiques de `70 MiB` :

```
# lvcreate -L 70MiB vg_disque
  Rounding up size to full physical extent 72,00 MiB
  Logical volume "lvol0" created.

# lvcreate -L 80MiB vg_disque
  Volume group "vg_disque" has insufficient free space (18 extents): 20 required.

# lvcreate -L 70MiB vg_disque
  Rounding up size to full physical extent 72,00 MiB
  Logical volume "lvol1" created.
```

On jette un oeil, les deux volumes logiques sont bien des périphériques en mode bloc :

```
# lsblk
NAME                MAJ:MIN RM   SIZE RO TYPE  MOUNTPOINT
...
loop0                 7:0    0   100M  0 loop
├─loop0p1           259:0    0    49M  0 part
│ └─vg_disque-lvol1 253:2    0    72M  0 lvm
└─loop0p2           259:1    0    50M  0 part
loop1                 7:1    0   100M  0 loop
├─vg_disque-lvol0   253:1    0    72M  0 lvm
└─vg_disque-lvol1   253:2    0    72M  0 lvm
```

On remarque que bien que `lsblk` nous montre la situation comme s'il s'agissait d'arbres, en fait `vg_disque-lvol1` apparaît 2 fois, car ce périphérique bloc est à cheval entre les périphériques `loop0p1`

et `loop1`. Comme on l'a discuté en cours, la relation entre les périphériques en mode bloc est plutôt un DAG (*directed acyclic graph*, structure représentant une relation de dépendance).

Notez aussi que les volumes physiques `LVM` et le volume group `LVM` ne sont pas des périphériques bloc, mais des étapes intérmédiaires spécifiques à `LVM` (ils n'apparaîssent pas dans la sortie de la commande `lsblk`).

On chiffre le premier volume logique :

```
# cryptsetup luksFormat /dev/mapper/vg_disque-lvol0

WARNING!
========
Cette action écrasera définitivement les données sur /dev/mapper/vg_disque-lvol0.

Are you sure? (Type uppercase yes): YES
Saisissez la phrase secrète pour /dev/mapper/vg_disque-lvol0 :
Vérifiez la phrase secrète :
```

Remarque, si on fait un :

```
# hexdump -C /dev/mapper/vg_disque-lvol0
```

on observe qu'un énorme header a été ajouté à la suite du header `LVM` : il commence par "`SKUL`" (qui est le renversé de "`LUKS`" : *Linux Unified Key Setup*), c'est le header de la nouvelle "couche de chiffrement" correspondant au périphérique bloc chiffré. Ce header contient de nombreux octets apparement aléatoires, car il contient la clef de chiffrement du périphérique chiffré, cette clef est elle-même chiffrée avec la phrase de passe que nous venons d'entrer.

Après le chiffrement de ce périphérique en mode bloc, on ne voit pas de périphérique en mode bloc apparaître :

```
# lsblk
NAME                MAJ:MIN RM   SIZE RO TYPE  MOUNTPOINT
loop0                 7:0    0   100M  0 loop
├─loop0p1           259:0    0    49M  0 part
│ └─vg_disque-lvol1 253:2    0    72M  0 lvm
└─loop0p2           259:1    0    50M  0 part
loop1                 7:1    0   100M  0 loop
├─vg_disque-lvol0   253:1    0    72M  0 lvm
└─vg_disque-lvol1   253:2    0    72M  0 lvm
```

C'est normal car, étant chiffré, on a un truc illisible, il faut d'abord permettre au système de le déchiffrer (en lui donnant la phrase de passe) pour que le système puisse le manipuler :

```
# cryptsetup open /dev/mapper/vg_disque-lvol0 crypt_disque
Saisissez la phrase secrète pour /dev/mapper/vg_disque-lvol0 :
```

On regarde qu'un périphérique en mode bloc est bien apparu :

```
# lsblk
NAME                MAJ:MIN RM   SIZE RO TYPE  MOUNTPOINT
loop0                 7:0    0   100M  0 loop
├─loop0p1           259:0    0    49M  0 part
│ └─vg_disque-lvol1 253:2    0    72M  0 lvm
└─loop0p2           259:1    0    50M  0 part
loop1                 7:1    0   100M  0 loop
├─vg_disque-lvol0   253:1    0    72M  0 lvm
│ └─crypt_disque    253:3    0    56M  0 crypt
└─vg_disque-lvol1   253:2    0    72M  0 lvm
```

Remarque : les périphériques bloc de type LVM2 et LUKS sont numérotés /dev/dm-<n> mais sont accessibles par leur petit nom dans /dev/mapper/ via un lien symbolique :

```
# ls -l /dev/mapper/
total 0
crw------- 1 root root 10, 236 mars   2 15:05 control
lrwxrwxrwx 1 root root       7 mars   3 00:19 crypt_disque -> ../dm-2
lrwxrwxrwx 1 root root       7 mars   3 00:19 vg_disque-lvol0 -> ../dm-0
lrwxrwxrwx 1 root root       7 mars   3 00:22 vg_disque-lvol1 -> ../dm-1
```

À partir de là, on peut installer des systèmes de fichiers dans les périphériques en mode bloc qu'on a obtenu, au hasard ext4, xfs, vfat :

```
# mkfs.ext4 /dev/mapper/crypt_disque
mke2fs 1.44.5 (15-Dec-2018)
Creating filesystem with 57344 1k blocks and 14336 inodes
Filesystem UUID: 36386848-5f90-4ab7-b7d8-ad1be16bfcbb
Superblock backups stored on blocks:
    8193, 24577, 40961

Allocating group tables: done
Writing inode tables: done
Creating journal (4096 blocks): done
Writing superblocks and filesystem accounting information: done


# mkfs.xfs /dev/mapper/vg_disque-lvol1
meta-data=/dev/mapper/vg_disque-lvol1 isize=512    agcount=4, agsize=4608 blks
         =                       sectsz=512   attr=2, projid32bit=1
         =                       crc=1        finobt=1, sparse=1, rmapbt=0
         =                       reflink=0
data     =                       bsize=4096   blocks=18432, imaxpct=25
         =                       sunit=0      swidth=0 blks
naming   =version 2              bsize=4096   ascii-ci=0, ftype=1
log      =internal log           bsize=4096   blocks=855, version=2
         =                       sectsz=512   sunit=0 blks, lazy-count=1
realtime =none                   extsz=4096   blocks=0, rtextents=0

# mkfs.vfat /dev/loop0p2
mkfs.fat 4.1 (2017-01-24)
```

On peut monter ces systèmes de fichier sur des répertoires :

```
# mkdir m1 m2 m3

# mount /dev/mapper/crypt_disque m1
# mount /dev/mapper/vg_disque-lvol1 m2
# mount /dev/loop0p2 m3

# mount
[...]
/dev/mapper/crypt_disque on /tmp/m1 type ext4 (rw,relatime)
/dev/mapper/vg_disque-lvol1 on /tmp/m2 type xfs (rw,relatime,attr2,inode64,noquota)
/dev/loop0p2 on /tmp/m3 type vfat (rw,relatime,fmask=0022,dmask=0022,codepage=437,iocharset=asc
```

Par contre, toutes ces encapsulations et systèmes de fichiers prennent de la place (tables des inoeuds, etc), de sorte qu'on a perdu par rapport aux périphériques en mode bloc initiaux :

```
# df -h
```

```
Sys. de fichiers            Taille Utilisé Dispo Uti% Monté sur
[...]
/dev/mapper/crypt_disque       51M    1,1M   46M   3% /tmp/m1
/dev/mapper/vg_disque-lvol1    69M    4,0M   65M   6% /tmp/m2
/dev/loop0p2                   50M       0   50M   0% /tmp/m3
```

On a bien un truc qui marche :

```
# echo hop > m1/plop
# ls m1/
lost+found  plop
```

Remarque : on peut obtenir des infos sur les périphériques bloc et les systèmes de fichiers avec d'autres outils :

On peut lister les périphériques bloc avec "`fdisk -l`" :

```
# fdisk -l
...
...
...

Disque /dev/loop0 : 100 MiB, 104857600 octets, 204800 secteurs
Unités : secteur de 1 × 512 = 512 octets
Taille de secteur (logique / physique) : 512 octets / 512 octets
taille d'E/S (minimale / optimale) : 512 octets / 512 octets
Type d'étiquette de disque : dos
Identifiant de disque : 0xf2c230bd

Périphérique Amorçage  Début    Fin Secteurs Taille Id Type
/dev/loop0p1            2048 102399   100352    49M 83 Linux
/dev/loop0p2          102400 204799   102400    50M 83 Linux


Disque /dev/loop1 : 100 MiB, 104857600 octets, 204800 secteurs
Unités : secteur de 1 × 512 = 512 octets
Taille de secteur (logique / physique) : 512 octets / 512 octets
taille d'E/S (minimale / optimale) : 512 octets / 512 octets


Disque /dev/mapper/vg_disque-lvol0 : 72 MiB, 75497472 octets, 147456 secteurs
Unités : secteur de 1 × 512 = 512 octets
Taille de secteur (logique / physique) : 512 octets / 512 octets
taille d'E/S (minimale / optimale) : 512 octets / 512 octets


Disque /dev/mapper/vg_disque-lvol1 : 72 MiB, 75497472 octets, 147456 secteurs
Unités : secteur de 1 × 512 = 512 octets
Taille de secteur (logique / physique) : 512 octets / 512 octets
taille d'E/S (minimale / optimale) : 512 octets / 512 octets


Disque /dev/mapper/crypt_disque : 56 MiB, 58720256 octets, 114688 secteurs
Unités : secteur de 1 × 512 = 512 octets
Taille de secteur (logique / physique) : 512 octets / 512 octets
taille d'E/S (minimale / optimale) : 512 octets / 512 octets
```

On peut directement regarder le fichier `/proc/partitions` :

```
# cat /proc/partitions
major minor  #blocks  name
...
...
...
   7        0     102400 loop0
 259        0      50176 loop0p1
 259        1      51200 loop0p2
   7        1     102400 loop1
 253        1      73728 dm-0
 253        2      73728 dm-1
 253        3      57344 dm-2
```

On peut regarder les systèmes de fichiers montés avec `findmnt`, qui est plus sympa à lire que ce que retourne `mount` :

```
# findmnt
TARGET                                SOURCE                      FSTYPE      OPTIONS
/                                     ......                      ......      .......
├─/...
├─/...
├─/...
└─/tmp                                tmpfs                       tmpfs       rw,relatime,s
  ├─/tmp/m1                           /dev/mapper/crypt_disque    ext4        rw,relatime
  ├─/tmp/m2                           /dev/mapper/vg_disque-lvol1 xfs         rw,relatime,a
  └─/tmp/m3                           /dev/loop0p2                vfat        rw,relatime,f
```

Si on veut tout ranger avant de quitter, on remonte le temps :

```
# umount m1 m2 m3

# cryptsetup close crypt_disque

# vgremove vg_disque
Do you really want to remove volume group "vg_disque" containing 2 logical volumes? [y/n]: y
Do you really want to remove active logical volume vg_disque/lvol0? [y/n]: y
  Logical volume "lvol0" successfully removed
Do you really want to remove active logical volume vg_disque/lvol1? [y/n]: y
  Logical volume "lvol1" successfully removed
  Volume group "vg_disque" successfully removed

# losetup -d /dev/loop0 /dev/loop1
```

Exercice : faites un dessin de la situation, avec les empilements de périphériques en mode bloc, les systèmes de fichiers, et leurs montages dans l'arborescence.