Devices, disks and filesystems

Grado en Informática 2022/2023
Departamento de Ciencias de la Computación
y Tecnologás de la Información
Facultad de Informática
Universidad de Coruña

Antonio Yáñez Izquierdo

Contenidos I

Contenidos II

Contenidos III

- Quota concepts
- linux
- BSD
- solaris

Devices, device files and device filesystems

Devices and device files

- to access devices unix systems use *device files*
  - a *device file* is a special type of file (either *block device* or *character device*) that contains the necessary information for the kernel to access the device, that is, which *device driver* to use and which unit inside that *device driver*. That's to say the *major* and *minor* numbers
- all device files are in the `/dev` directory. The name of the devices depends on the UNIX variant we are using
- in addition to the *device driver* the kernel **needs** the device files to access the device
- device files can be created with the '`mknod`' command, provided we already know both the major and minor numbers

/dev directory. Traditional approach

- the /dev directory is populated with the appropriate device files during O.S. installation
- should a new device file be needed (for example, new hardware has been added) its device file must be manually created with the 'mknod' command
- a shell script (or program) called MAKEDEV is usually located in this directory to help us create the device files (it contains the suitable major and minor numbers)
- Current OpenBSD release (7.0) still uses this approach
- Current NetBSD (9.0) also uses this approach for devices other than the virtual terminals

Devices, device files and device filesystems
→device filesystems

device filesystems

- another approach is that the kernel creates a virtual device filesystem (analog to the `/proc` filesystem) populated with all the devices detected during boot and it mounts it on `/dev`
- this filesystem is of type *devfs* in FreeBSD, *devtmpfs* in linux and *devfs* in solaris
- the command `'devfsadm'` manages the device filesystems in solaris (logical devices under `/dev` linked to the physical devices under `/devices`)

device filesystems

- in FreeBSD we could mount the device filesystem in '/target' with

```
mount -t devfs devfs /target
```

- the same could be done in linux with

```
# mount -t devtmpfs devfs /target
```

- NetBSD uses a filesystem of type *ptyfs* (mounted on `/dev/pts`) for the virtual terminals

Adding support for devices. Kernel modules

kernel modules

- most present unix like kernels are modular, that is, they support modules
  - a module is a piece of code that can be loaded or unloaded from a running kernel, usually adding support to some feature or providing a device driver for some devices
- the administrator usually needs not care about module loading and unloading as the kernel takes care itself. Sometimes, in special occasions, such as when the kernel does not correctly identify a device, or when we want to force the unload of some module we can use utilities to do so.

kernel modules

Solaris 'modinfo' to get info on the loaded modules, 'modload' to load a module and 'modunload' to unload a module

Linux 'lsmod' to get info on the loaded modules, 'insmod' and 'modprobe' to load a module and 'rmmod' to unload a module

FreeBSD 'kldstat' to get info on the loaded modules, 'kldload' to load a module and 'kldunload' to unload a module

NetBSD 'modstat' to get info on the loaded modules, 'modload' to load a module and 'modunload' to unload a module

OpenBSD does not support kernel modules

kernel modules

- support for devices not present in the running kernel can be added via a module
- modules can be supplied either
  - as a binary file, loadable with one of the utilities seen before. This is the form when dealing with propietary drivers
  - source file to be compiled (usually needing the kernel source tree or at least its headers) and loaded afterwards
- modules could present a security risk. That the reason why OpenBSD has dropped support for modules

Adding support for devices. Kernel modules
→Tuning kernel parameters

Tuning kernel parameters

- some behaviour of the kernel can be fine tuned
  - **Solaris** Through `/etc/system` and some files in `/etc/default`
  - ***BSD** With the 'sysctl' command (file `/etc/sysctl.conf` for boot time configuration)
  - **linux** It used to be through the `/proc` filesystem. Nowadays the recommended method is to the command 'sysctl' and the file `/etc/sysctl.conf` for boot time configuration

Organisation of the UNIX file system

Organisation of the UNIX file system

- although the UNIX file system looks like a tree (with different file system *mounted* onto different directories) it is actually a graph, as links (both real and symbolic) can be created
- its organization is pretty much standard through different flavors of UNIXes
- the *boot loader* needs to know only the *root* file system. All the file systems to be mounted at boot time can be specified through the `/etc/fstab` file
  - file (`/etc/vfstab` in Solaris)
- we'll describe briefly what we can find in the different top level directories of a typical installation

Organisation of the UNIX file system

- `/bin` Essential system binaries
- `/boot` Files needed to boot the system, kernel, initial ram disks, sometimes modules, loader related stuff (on EFI systems the Efi System Partition is usually mounted in `/boot/efi`) ...
- `/dev` Files for accessing the devices (Solaris has also the `/devices` directories for the *physical* devices)
- `/etc` System administration: configuration files
- `/home` User's home directories (on Solaris systems this would be `/export/home`)

Organisation of the UNIX file system

- `/lib` Libraries
- `/media` Mount points for removable media
- `/mnt` Mount point for temporary mounts
- `/opt` Optional system software
- `/proc` The `/proc` file system mount point

Organisation of the UNIX file system

- `/root` The home directory for the system administrator
- `/sbin` Essential administrator binaries
- `/sys` System files and filesystems (`/system` in Solaris)
- `/tmp` World writable directory for creating temporary files for users and applications. Gets cleaned at each boot

Organisation of the UNIX file system

- `/usr` The bulk of the installed software and applications.
  - Usually holds all of the system software except for the base (bare) system
  - Subdirectories for executable binaries, libraries, graphic environment, man pages ...
  - It also holds all the documentation
- `/var` System administration: Dynamic stuff. Logs, databases, spool directories, pids ...

Managing disks. Partitioning schemes

Managing discs

- We'll see the two approaches to using disks
- **Traditional approach**
  - We *partition* the disks
  - We create a file system in a partition
  - We mount filesystems onto different directories to create a directory structure
- **LVM**
  - We create Physical Volumes on discs and partitions and combine them to create Volume Groups
  - We create Logical Volumes on top of the Volume Groups
  - We create file systems onto the Logical Volumes and mount them onto different directories to create a directory structure

Managing disks. Partitioning schemes
→Discs, partitions and filesystems

Discs and partitions

- in UNIX system the traditional approach is to create filesystems on block devices to store our information in files
- this devices are usually *partitioned*, and we use the partitions instead the whole devices. Most common partition formats are the MBR and GPT formats. BSD system use the disklabel format, ad Solaris uses the VTOC format (which is a variation on the disklabel format)
- progams to partition disk are `fdisk`, `cfdisk`, `parted`, `gpart`, `format`, `bsdlabel`, `disklabel` ...
- older devices, such as floppy discs (typically named /dev/fd0, /dev/fd1 ...) were used without partitioning. We still can do that to usb pendrives and memory cards although it is not the usual method

Partition schemes and device naming

Partition and namimg schemes

- Different unixes treat partitions differently and, in fact, they use different partition schemes.
- we'll consider how the different *unix* flavors treat the different partition schemes (MBR, GPT, disklabel . . .)
- partition schemes, MBR, GPT, BSD disklabel and solaris' VTOC are discussed in lesson 2
- formatting and accessing the devices is done using the device files, which have a name

Partition schemes and device naming
→linux

linux partition and namimg schemes

- disks are named `sda, sdb, sdc ....` So the device files for disks are `/dev/sda, /dev/sdb ...`
- (older sistems used hda, hdb, hdc . . . for IDE hard disks)
- linux uses both MBR and GPT, and understands other partition schemes.
- BIOS systems must boot from disks with MBR partitions and UEFI systems boot from files on the Efi System Partition on a GPT partitioned disk

linux partition and namimg schemes

- GPT partitions are numbered, `sda1`, `sda2`, `sda3` ..., so the device files are `/dev/sda1`, `/dev/sda2`, ...
- MBR primary partitions are numbered, `sda1`, `sda2`, `ada3`, `ada4`. Should the system have an *extended partition*, the *logical units* or *secondary partitions* inside the *extended partition* are numbered from 5 onwards: `/dev/sda5`, `/dev/sda6`, `/dev/sda7` ...

Partition schemes and device naming →freeBSD

FreeBSD partition and namimg schemes

- ATA (both serial and parallel) disks are named ada0, ada1, ada2 . . . . Non ATA disks (such as SCSI) are named da0, da1 . . .
- freeBSD uses GPT partitions on UEFI systems and disklabel on BIOS systems
- on UEFI systems partitions are labeled p1, p2 . . . . So the device files for GPT partitions on the first disk would be `/dev/ada0p1, /dev/ada0p2, /dev/ada0p3 ....`

FreeBSD partition and namimg schemes

- '`gpart ada0`' would access the GPT partition table on the first disk
- on BIOS systems freeBSD uses *disklabel* (accessible through the `bsdlabel` command). FreeBSD's disklabel can hold up to 8 partitions, which are labeled a, b, c . . . . *a* is the *root* file system, *b* the swap and *c* represents the whole disk/partition.

FreeBSD partition and namimg schemes I

- Two ways two use the *disklabel* on BIOS systems
  - the disk is partitioned with *disklabel*,
    - partitions are labeled /dev/ada0a, /dev/ada0b, /dev/ada0c ...
    - `bsdlabel -e ada0` we would be able to access and edit the disklabel
  - the disks is partitioned with MBR and the disklabel is created inside an MBR partition or *slice* (with id a5).
    - Primary MBR partitions (on the first disk) are named ada0s1, ada0s2, ada0s3 and ada0s4 and can be accessed with `/dev/ada0s1, /dev/ada0s2 /dev/ada0s3 and /dev/ada0s4`

FreeBSD partition and namimg schemes II

- if the primary MPR partition containing the disklabel is ada0s2, the disklabel partition device files would be `/dev/ada0s2a, /dev/ada0s2b, /dev/ada0s2c ....` In this case
- `fdisk ada0` would access the MBR partition table
- `bsdlabel -e ada0s2` would edit the disklabel

Partition schemes and device naming →OpenBSD

OpenBSD partition and namimg schemes

- disks are named `wd0, wd1, wd2 ...` (IDE disks) or `sd0, sd1, sd2, sd3` (sata or scsi disks)
- OpenBSD understands GPT and MBR partitions but does not use them
- it creates its own disklabel inside one of the GPT or MBR partitions
- its disklabel can hold up to 16 partitions ('a'..'p') and does not have the 2TB limit as MBR does. 'a' must be the root, 'b' the swap and 'c' represents the whole disk

OpenBSD partition and namimg schemes

- partitions inside the disklabel are named `/dev/wd0a`, `/dev/wd0b`, `/dev/wd0d ...` or `/dev/sd0a`, `/dev/sd0b ...` depending on the disk type
- MBR and GPT partitions not on the disklabel cannot be accessed and thus they aren't named: if we want to accesss another GPT or MBR partition we have to include it in the disklabel
- `fdisk -e sd0` to edit the MBR or GPT partition tables
- `disklabel -e sd0` to edit the disklabel

Partition schemes and device naming →NetBSD

NetBSD partition and namimg schemes

- As in OpenBSD disks are named `wd0, wd1, wd2 ...` or `sd0, sd1, sd2, sd3` depending on the connection
- NetBSD understands GPT and MBR
- When using MBR, it creates its own disklabel inside one of the MBR partitions.
- its disklabel can hold up to 16 partitions but uses a 32 bit number for the partition size as MBR does. 'a' must be the root, 'b' the swap and 'c' represents the NetBSD part of the disk and 'd' the whole disk
- partitions inside the disklabel are named `/dev/wd0a, /dev/wd0b, /dev/wd0d ...` or `/dev/sd0a, /dev/sd0b ...` depending on the disk type

NetBSD partition and namimg schemes

- When using GPT partition table, NetBSD uses the partitions provided by GPT
- In that case NetBSD uses a unified partition interfaz (*wedges*) and partitions ar named `/dev/dk0, /dev/dk1 ...`
- `fdisk` to access the MBR partition table
- `disklabel` to access the disklabel
- `gpt` to access the GPT partition table
- `dkctl` to access the disk with *wedges*

Partition schemes and device naming
→solaris

solaris partition and namimg schemes

- Solaris disks are named `/dev/dsk/cXtYdZ`, where X,Y,X are numbers depend on the hardware connection between the disk and the system. Sometimes disks are named /dev/dsk/cXdY.
- Solaris supports two type of partitions: VTOC (disklabel) partitions and GPT partitions.
- VTOC partitions are named with the s (for slice) and the partition number. So the first partition on disk `c1d0t0` would be `/dev/dsk/c1d0t0s0`

solaris partition and namimg schemes

- MBR primary partitions are numbered p1,p2,p3,p4.(solaris does not use extended partitions). p0 stands for the partition table itsef,
  - should we want to use `'fdisk'` directly instead on using it from the `'format'` utility, we would do `'fdisk /dev/rdsk/c0t2d0p0'`
- when in a MBR scheme, the VTOC is created inside one of the MBR primary partitions. On the other hand, GPT partitions are used directly
- those are the names of the *logical* devices which are a symbolic link to the physical device under `/devices`

solaris partition and namimg schemes

- solaris 10 cannot boot from uefi firmware but can (in its 64 bit version) use GPT partitioned disks
- solaris 11 can boot from both BIOS and UEFI firmware and can use both VTOC and GPT labeled disks
- the `'format'` utility allows us to access the MPR partitions through its 'fdisk' menu, and the VTOC partitions through its 'partition' menu
- if we want to use the `fdisk` utility directly we'd do `'fdisk/dev/rdsk/c0t2d0p0'`
- we use `format -e` to access GPT labeled disks

Partition schemes and device naming
→disk devices and raw disk devices

disk devices and raw disk devices

- disk and partition devices are block devices
- some Operating Systems, such as OpenBSD and solaris, have a separate character device for each disk and partition device
- it is called the *'raw'* device and is used in some operations, for example, checking or creating the file system
- in OpenBSD and NetBSD they are named with an `'r'` before the name of the actual device: `/dev/rsd0a`, `/dev/rwd0c` ...
- in solaris, their name is the same as the block device but instead of being found under `/dev/dsk`, they are located in the `/dev/rdsk` directory: `/dev/rdsk/c1d0t0s0`, `/dev/rdsk/c1d0t0s7` ...

Creating and accesing filesystems

Creating and accesing filesystems
→creating filesystems

creating filesystems

- different UNIX variants use and support different file system types.
- the most common used filesystems are ext2, ext3, ext4 (linux) ffs (BSD) ufs (solaris take on BSD's ffs) and ZFS (freeBSD and solaris)
- most unixes understand several file systems, for example FAT file system is understood by all

creating filesystems

- to create a file system on a device we use the command `'mkfs'` or `'newfs'` and the device file where we want to create that file system
- this commands takes arguments to indicate, the type of filesystem, and parameters to the file system creation: block size, fragment size, inodes per block . . .
- after creating the file system we can access it with the `mount` command

steps for adding a disk and creating a filesystem in it I

1. Physically adding the disk
2. Create the appropriate device files, most of the times we need do nothing as the kernel takes care of creating the devices itself
    - In OpenBSD and NetBSD we might need to create the device files with `MAKEDEV`
    - In Solaris we might need to do `'touch /reconfigure'` before rebooting to make the kernel check for new devices the next reboot and create the appropriate device files
3. Partition the drive (`fdisk`, `gpart`, `disklabel`, `bsdlabel`, `gpt`, `format ...`)

steps for adding a disk and creating a filesystem in it II

- In Solaris, OpenBSD and FreeBSD on MBR systems we have to first create an MBR partition and then a disklabel (or VTOC) in the MBR partition
- In OpenBSD on GPT system we first create a GPT partition and then a disklabel in the GPT BSD partition

4. Create the filesystem with '`mkfs`' or '`newfs`'
5. Access the filesystem with '`mount`'

Creating and accesing filesystems
→accessing filesystems

accessing filesystems

- before accessing a file system we have to place it (*mount it*) somewhere in the system's directory structure
- we use the command 'mount'. The following command places the file system in /dev/dsk/c0tdd0s5 onto directory /var

```
# mount /dev/dsk/c0tdd0s5 /var
```

- the 'mount' command uses multiple arguments to specify the file system type, and mount options such as *read-only, nosuid, noexec, noatime ...*
- user administrator privileges are required to mount filesystems

accessing filesystems

- the following examples mount a windows FAT filesystem in the second disk's third primary MBR partition onto directory `/win`
- linux: `mount -t vfat /dev/sdb3 /win`
- FreeBSD: `mount -t msdosfs /dev/ada1s3 /win`
- solaris: `mount -F pcfs /dev/dsk/c0t1d0p3 /win`
- OpenBSD: `mount -t msdos /dev/wd1i /win` (we'll assume to have used the 'i' entry on OpenBSD's disklabel)
- NetBSD: `mount -t msdos /dev/wd1i /win` (we'll assume to have used the 'i' entry on NetBSD's disklabel)

accessing filesystems

- if it were the third GPT partition on the second disk
- linux: `mount -t vfat /dev/sdb3 /win`
- FreeBSD: `mount -t msdosfs /dev/ada1p3 /win`
- solaris: `mount -F pcfs /dev/dsk/c0t1d0s2 /win`
- OpenBSD: `mount -t msdos /dev/wd1i /win` (again we'll assume to have used the 'i' entry on openBSD's disklabel)
- NetBSD: `mount -t msdos /dev/dk6 /win` (we'll asume dk6 is the *wedge* of the windows FAT partition)

Creating and accesing filesystems

→the /etc/fstab file

the /etc/fstab file

- Now that we know how to place different file systems on different directories. How do we create the complete system's directory structure at boot time?
- The root file system (that is, the file system whose '/' directory is the system's '/') is specified via the boot loader at boot time
- The file /etc/fstab contains the file systems to be mounted at boot time
- This file is processed when the system boots and the file systems in it are mounted before the system is ready

format of the /etc/fstab file

- each line of the /etc/fstab file has the following fields

device device to mount

dir directory to mount onto

type type of filesystem

opts comma separated list of mount options (filesystem type dependant)

dump 1 or 0 depending whether the filesystem backup is controlled by the *dump* command

pass specifies if the device is checked at boot time

format of the /etc/fstab file

- the most usual mount options are *defaults, ro, rw, nosuid, nexec, noauto, usrquota ...*
- example of /etc/fstab file in linux

```
 # <file system> <mount point> <type> <options> <dump> <pass>
/dev/sda1          /           ext4    defaults      0      1
/dev/sda7          none        swap      sw          0      0
/dev/sda5          /var        xfs     noexec,nosuid 0      1
/dev/cdrom         /cdrom      iso9660 defaults,ro,user,noauto 0   0
```

uuids in /etc/fstab in linux

- in modern linux systems, we can, instead of using the name of the device (i.e. /dev/sda4), use the UUID (Universally Unique IDentifier)
- so, an entry in the `/etc/fstab` file will look like this

```
UUID="d39577d4-fb9b-4b18-be4e-53ff32dbf856" /home ext4 noatime  0   2
```

- that number can be obtained with the command **blkid**. Example

```
root@abyecto:/home/antonio# blkid /dev/sdb2
/dev/sdb2: UUID="7b127a41-0ff1-45ed-8c0a-dac6816cd02c" TYPE="ext2" PARTUUID="e9ddd7cb-911f-3b47-916b-d7c9
root@abyecto:/home/antonio#
```

duids in /etc/fstab in openBSD

- Each time we use disklabel on a disk, the `'disklabel'` utility generates a unique identifier for the disk *duid*
- the disk can be then referenced by that identifier and so can the partitions

```
# disklabel wd0
# /dev/rwd0c:
type: ESDI
disk: ESDI/IDE disk
label: VBOX HARDDISK
duid: 8d0c71fb057cdd39
.......
........
16 partitions:
#                size           offset  fstype [fsize bsize   cpg]
  a:         33000000               64  4.2BSD   2048 16384 12958 # /
  b:           543656         33000064    swap                    # none
  c:         33554432                0  unused
# cat /etc/fstab
8d0c71fb057cdd39.b none swap sw
8d0c71fb057cdd39.a / ffs rw,wxallowed 1 1
#
```

names in /etc/fstab in FreeBSD

- in FreeBSD we can assign a label (name) to a partition with the command `tunefs -L label partition`. Example

  `# tunefs -L datillos  /dev/ada0s2a`

- afterwards we could use the label *'datillos'* in the `/etc/fstab` file
- this labels are shown in `/dev/ufs`
- should we want to use an UUID we can generate it with the command `'uuidgen'` and assign it to the partition

  ``# tunefs -L `uuidgen`  /dev/ada0s2a``

Creating and accesing filesystems
→the /etc/vfstab file

the /etc/vfstab file

- solaris systems have an /etc/vfstab file instead of the traditional /etc/fstab file
- its format is very similar to the /etc/fstab file. One line for each file system to be mounted, with fields separated by blanks. The fields in each line are:
  - device to mount
  - device to fsck
  - mount point
  - File Sydtem type
  - fsck pass
  - mount at boot
  - mount options

the /etc/vfstab file

- Example of a /etc/vfstab file in solaris

```
#device          device          mount           FS      fsck    mount   mount
#to mount        to fsck         point           type    pass    at boot options
#
fd       -       /dev/fd fd      -       no      -
/proc    -       /proc   proc    -       no      -
/dev/dsk/c0t0d0s1        -       -       swap    -       no      -
/dev/dsk/c0t0d0s0        /dev/rdsk/c0t0d0s0      /       ufs     1       no      -
/dev/dsk/c0t0d0s7        /dev/rdsk/c0t0d0s7      /export/home    ufs     2       yes     -
/devices         -       /devices        devfs   -       no      -
sharefs -        /etc/dfs/sharetab       sharefs -       no      -
ctfs     -       /system/contract        ctfs    -       no      -
objfs    -       /system/object  objfs   -       no      -
swap     -       /tmp    tmpfs   -       yes
```

Loopback devices

Loopback devices

- some times it is useful to access a file as if it were a block device
  - we'd like to create a file system on a file and try it before writing it to a device
  - we want to try some software the works on a file system and we do not have a spare file system
  - we want to mount an image file from a file system (for example an iso cdimage) without having to burn the actual media
  - ...
- this is done by what we call the loopback device

Loopback devices: linux

- in linux this is just done with an option to the mount command. The option 'loop'
- in the following examples we'll like to mount an iso image named 'cdimage.iso' which is at /home/user01'

```
# mount -t iso9660 -o loop /home/user01/cdimage.iso /mnt
```

- and after we are done with using the image

```
# umount /mnt
```

Loopback devices: solaris

- we create and destroy the loopback devices with `lofiadm`. When creating the device this command reports the device name

```
# lofiadm -a /home/user01/cdimage.iso
/dev/lofi/1
# mount -F hsfs /dev/lofi/1 /mnt
```

- and after we are done, we umount the image and destroy the device

```
# umount /mnt
# lofiadm -d /dev/lofi/1
```

Loopback devices: freeBSD

- we use the -f (file) option of `mdconfig`. This will inform of the device name

```
# mdconfig -d  /home/user01/cdimage.iso
md0
# mount -t cd9660 /dev/md0 /mnt
```

- and after we are done, we umount the image and destroy the device

```
# umount /mnt
# mdconfig -d -u 0
```

Loopback devices: OpenBSD and NetBSD

- we create a link with a *vnd* device and the file with `vnconfig`.

```
# vnconfig vnd0  /home/user01/cdimage.iso
md0
# mount -t cd9660 /dev/vnd0c /mnt
```

- and after we are done, we umount the image and unlink the device

```
# umount /mnt
# vnconfig -u vnd0
```

Read only file systems

Read only fylesystems

- Read only filesystems are file systems that once created cannot be written to
- they are typically used in WORM m (Write Once Read Many) media, such as CDs ad DVDs
- they are also used on embedded systems
- the two mostly widespread read only filesystem in the UNIX world are *iso* and *squashfs*

Read only file systems
→ISO file system

ISO filesystem

- Also known as ISO9660, cd9660 or HighSierra
- it is the filesystem of choice for optical WORM media
- Directories and files are stored as sequential series of sectors
- File names valid characters are upper case letters, digits, '_', and ONE dot
- Directories can have up to eight levels depth
- Filenames can have up to 8 character names (and a 3 character extension (level 1) or 31 character length (level 3, which also allows a file to be stored as a non contiguous set of extents)
- It uses a 32 bit integer for the size, so maximum file size is 4Gb

ISO filesystem extensions

- To overcome the limitations of the standard, several extensions were developed
- **joliet**
    - developed by Microsoft for Windows O.S. (starting on windows 95)
    - file names can exceed 100 characters in length
    - supports *Unicode* characters

ISO filesystem extensions

- **rock ridge**
  - for UNIX like systems
  - longer file names (up to 255)
  - support for lower case characters
  - unix style file permissions, uids, gids and timestamps
  - support for symbolic links
  - deeper (more than 8 levels) directory hierarchy

Accessing ISO file systems

- ISO file systems are supported *out of the box* by all present unix-like systems
- Optical media can be mounted with the '*mount*' command, although the name of the filesystem varies from one system to another
  - *iso9660* on linux `mount -t iso9660 ...`
  - *cd9660* on BSD systems `mount -t cd9660 ...`
  - *hsfs* on Solaris `mount -F hsfs ...`
- We can mount files containing an ISO filesystem (usually referred as iso files, by using the loopback devices as seen in previous section

Creating an ISO file systems

- ISO filesystems can be created (onto a file) with the `mkisofs` utility (available in all present unix systems, sometimes as part of the *cdrtools* software package)
- this iso image could be written to optical media with the `cdrecord` program.
- we can also use the utility `growisofs` (part of the *dvd+rw-tools* software package to append sessions to an optical media
- File systems on optical media can also be created with a variety of graphical mastering programs (k3b, brasero . . .)

UDF filesystem

- Stands for Universal Disk Format
- It's the filesystem usually used in DVD a newer optical media
- It allows for creating, deleting and resizing files in optical RW media with *packet writing*
- All present UNIX systems allow mounting udf media using the `udf` mount option

Read only file systems
→squashfs filesystem

squashfs filesystem

- read only filesystem originally developed for linux
- filesystem with compressed inodes, files and directories, allows for links and devices on the file system
- maximum file size si $2^{64} - 1$
- usually used for embedded systems and live operating systems
- it is also used for *snap* packages
- initially used *gzip* compression. Presently, it supports other compression algorithms such as LZMA, LZMA2, xz ...

squashfs filesystem

- block size ranges form 64kb to 1Mb, 128Kb by default, to achieve grater compression ratio
- has *fragments* blocks. A block can contain several files smaller than one block
- duplicate files are removed (when created with the adequate option
- uids and gids are stored in the filesystems, as well as permissions and timestamps

accessing squashfs filesystem: linux

- we can create an squashfs filesystem with the `mksquashfs` utility
- we can get files from a squashfs filesystem with the `unsquashfs` utility
- `mksquashfs` and `unsquashfs` are part of the *squashfs-tools* software package
- we can mount a `unsquashfs` with the `-t squashfs` option to mount

```
antonio@abyecto:~$ mksquashfs FSO-Docs/ Downloads/ prueba.squash
antonio@abyecto:~$ su
Password:
root@abyecto:/home/antonio# mount -t squashfs ./prueba.squash /mnt/
root@abyecto:/home/antonio# ls -l /mnt/
total 0
drwxr-xr-x 5 antonio antonio 646 Mar 19 11:11 Downloads
drwxr-xr-x 2 antonio antonio 143 Oct 15  2018 FSO-Docs
```

accessing squashfs filesystem: FreeBSD

- we can create an squashfs filesystem with the `mksquashfs` utility
- we can get files from a squashfs filesystem with the `unsquashfs` utility
- `mksquashfs` and `unsquashfs` are part of the *squashfs-tools* software package

accessing squashfs filesystem: FreeBSD

- to mount a `unsquashfs` we have to have the *fusefs-squashfuse* package installed and the have the `fusefs` kernel module loaded (or have the line `'fusefs_load=YES'` added to `/boot/loader.conf`)

```
[antonio@aso2 ~/Desktop]$ mksquashfs /bin /etc prueba.squash
antonio@aso2 ~/Desktop]$ su
Password:
root@aso2:/home/antonio/Desktop # squashfuse ./prueba.squash /mnt
fuse: failed to open fuse device: No such file or directory
root@aso2:/home/antonio/Desktop # kldload fusefs
oot@aso2:/home/antonio/Desktop # squashfuse ./prueba.squash /mnt
root@aso2:/home/antonio/Desktop # ls /mnt
bin etc
```

accessing squashfs filesystem: OpenBSD and NetBSD

- we can create an squashfs filesystem with the `mksquashfs` utility
- we can get files from a squashfs filesystem with the `unsquashfs` utility
- `mksquashfs` and `unsquashfs` are part of the *squashfs-tools* software package

Disk quotas

Disk quotas
→Quota concepts

Quotas

- *quotas* allow us to restrict the amount of space a user (or a group) can use on a file system
- *quotas* configuration is per user (or group) and filesystem
- *quotas* reside in the files `aquota.user` (or `aquota.group`) in the root directory of the filesystem where the quota is established

Quotas

- for each user (or group) in a file system we can establish a limit both on the files (*inodes*) and blocks (*space*) that a user or group can use. This is what we call the quota
- for each user (or group) in a file system both a soft and a hard limit (for both files and blocks) are configured
  - upon reaching the soft limit a warning is issued, but the write system calls still work
  - upon reaching the hard limit write system calls fail, so the user (or group) can never exceed the hard limit
  - the user (or group) can stay over the soft limit for a period of time (call *grace period*) after which the soft limit becomes the hard limit (write system calls fail)

Quota utilities

- most systems include a set of utilities to help manipulate the quotas
    - **quotacheck** creates, checks and/or repairs quota files in a files system
    - **quotaon, quotaoff** turns on (or off) quotas on a files system
    - **edquota** allows modification of a user (or group) quotas
    - **repquota, quota** reports the status of the quotas in a file system

Disk quotas
→linux

Enabling quotas

- to enable *quotas* on a file system we need
  1. have *quota* support in the kernel (mostly all preconfigured distro kernels come with quota support)
  2. mount the file system with the *usrquota* (and/or *grpquota*) option
  3. have installed the corresponding *quota* management programs (in debian type distros `apt-get install quota`)

Enabling quotas: utilities

- the *quota* package includes the following utilities
  - **quotacheck** creates, checks and/or repairs quota files in a files system
  - **quotaon, quotaoff** turns on (or off) quotas on a files system
  - **edquota** allows modification of a user (or group) quotas
  - **repquota, quota** reports the status of the quotas in a file system

Defining quotas: edquota

- we use the program **edquota** to establish quotas for different users. A summary of its usage is
    - **edquota -u name** opens the editor defined in $EDITOR for us to modify the soft and hard limits for user *name*.
    - **edquota -g grpname** opens the editor defined in $EDITOR for us to modify the soft and hard limits for group *grpname*.
    - **edquota -p prototype name** establishes quotas for user *name* the same as user *prototype*.
    - **edquota -t** establishes the grace period

Defining and stablishing quotas: example

- in the following example
  - we'll create the quota files (for both user and group) in the filesystem at `/dev/sda4`
  - we'll establish quotas in the filesystem at `/dev/sda4` for user *antonio* and group *bin*
  - we'll turn on quotas for that file system
  - we'll make every user defined locally in the system with the /bin/bash as his/her login shell have the same quota as user *antonio*
- note that the next time we boot the system, if the filesystem is mounted with the quota options on `/etc/fstab` the booting scripts will take care of checking and turning the quotas on, so we need do nothing

Defining and stablishing quotas: example

```
.
root@hardeningB:/home/antonio#
root@hardeningB:/home/antonio# mount -t ext4 -o usrquota,grpquota /dev/sda4  /datos/
root@hardeningB:/home/antonio#
root@hardeningB:/home/antonio#
root@hardeningB:/home/antonio# quotacheck -uv /datos/
quotacheck: Your kernel probably supports journaled quota but you are not using it. Consider switching to
quotacheck: Scanning /dev/sda4 [/datos] done
.....
quotacheck: Old file not found.
root@hardeningB:/home/antonio#
root@hardeningB:/home/antonio# quotacheck -gv /datos/
quotacheck: Your kernel probably supports journaled quota but you are not using it. Consider switching to
quotacheck: Scanning /dev/sda4 [/datos] done
quotacheck: Checked 2 directories and 2 files
quotacheck: Old file not found.
root@hardeningB:/home/antonio#
root@hardeningB:/home/antonio# quotaon /dev/sda4
root@hardeningB:/home/antonio#
root@hardeningB:/home/antonio# edquota -u antonio
root@hardeningB:/home/antonio# edquota -g bin
root@hardeningB:/home/antonio# edquota -t
root@hardeningB:/home/antonio# for name in `cat /etc/passwd | grep /bin/bash | cut -f1 -d:` ;
do  edquota -p antonio $name ; done
```

Defining and stablishing quotas: reboot

- in the last example, we saw how to define and stablish quotas on the filesystem in `/dev/sda4`
- if we want to continue to use quotas after the next reboot, we can do so in two ways
  1. remounting /dev/sda4 with the options `usrquota` and/or `grpquota` and manually running `quotacheck` and `quotaon`
  2. modify `/etc/fstab` adding the options `usrquota` and/or `grpquota` and let the initialization scrips take care of running `quotacheck` and `quotaon`
     - these initialization scripts are part of the *quota* package
     - we can control whether these scripts are going to be run at boot with `insserv`, `systemctl`, `chkconfig` . . . , depending on our linux distribution

Disk quotas
→BSD

Quotas in BSD

- in BSD the quotas are part of the base system so we need not install anything
- setup is pretty much the same as in linux except for two tiny details
  1. the mount options are named *userquota* and *groupquota* instead of *usrquota* and *grpquota*
  2. the quota files are named *quota.user* and *quota.group*
- the utilities operate as in linux. Obviously, the names of the devices are different.

Enabling quotas at boot in BSD

- **FreeBSD**
  - we add `quota_enable="YES"` to `/etc/rc.conf`.
  - we can speed up the booting of the system, sacrificing the checking of quotas on the disk with `check_quotas="NO"` in `/etc/rc.conf`, although this is not recommended
- **OpenBSD**
  - we add `check_quotas="YES"` to `/etc/rc.conf.local`.
  - if we want to speed up the booting of the system, sacrificing the checking of quotas on the disk (not recommended) instead of the aforementioned line in `/etc/rc.conf.local`, we simply add `'quotaon -a'` to `/etc/rc.local`
- **NetBSD**
  - we add `quota="YES"` to `/etc/rc.conf`.
  - this enables and checks quotas at boot

Disk quotas
→solaris

Quotas in Solaris: UFS

- Solaris's UFS supports quotas for users
- The utilities *quotacheck*, *edquota*, *repquota* and *quotaon* operate as in the other O.S.
- The filesystem in which we want to enable quota mas be mounted with the option `'rq'` (file `/etc/vfstab`)
- the file `'quota'` must be manually created in the root directory of the filesystem where we want to enable quotas, made administrator owned with permissions 600

```
# cd /DirectoryFileSystemIsMountedOn
# touch quota
# chmod 600
```

- Now we can edit the quotas for different users con `edquota`