Filesystems – Block file systems

Objective: configure and boot an embedded Linux system relying on block storage.

After this lab, you will be able to

- Manage partitions on block storage.
- Produce file system images.
- Configure the kernel to use these file systems
- Use the tmpfs file system to store temporary files

Goals

After doing the “A tiny embedded system” lab, we are going to copy the filesystem contents to a USB flash drive provided by the instructor. The filesystem will be split into several partitions, and we will boot the CALAO board with this USB disk, without using NFS any more. At the end of the lab, the filesystem layout should be :

- / mounted as a SquashFS filesystem, read-only, containing all the applications, libraries and configuration files for our system
- /www/upload/files mounted as an ext2 filesystem, read-write, to host the files uploaded to our Web server and the Web server logs.

Follow the steps below to set up such a layout.

Setup

Go to `/home/<user>/felabs/sysdev/fs`.

Reuse the kernel that you used in `/home/<user>/felabs/sysdev/tinysystem`.

If you didn't do or complete the `tinysystem` lab, you can use the `data/rootfs` directory instead.

Recompile it with support for `SquashFS` and `ext3`.

Boot your board with this new kernel and on the NFS filesystem you used in this previous lab.

USB disk partitioning

`fdisk` allows to partition block devices such as hard drives or USB flash drives. Partitioning consists of creating a descriptor table containing information on the different partitions on the device.

Using `fdisk` **on your workstation**, you will have 2 partitions on the USB flash drive provided by your instructor. The first one will be used to store the root file system and does not require more than 1MB. The second partition will handle the target local storage and will the use all the remaining space on the USB disk.

First find how the kernel has identified the USB disk, consult the kernel log:

```
dmesg
```

connect the key then look at the log

```
sd 3:0:0:0: [sdb] Attached SCSI removable disk
```

The block device representing the USB disk is seen as a SCSI drive mapped to `/dev/sdb`. If the auto-mounter opens the partitions already contained on the USB key, unmount each of them to avoid

conflicts.

In a terminal open the block device with `fdisk` :

```
$ sudo fdisk /dev/sdb
```

Display the on-line help by pressing the `m` key:

```
Command (m for help): m
Command action
   a   toggle a bootable flag
   b   edit bsd disklabel
   c   toggle the dos compatibility flag
   d   delete a partition
   l   list known partition types
   m   print this menu
   n   add a new partition
   o   create a new empty DOS partition table
   p   print the partition table
   q   quit without saving changes
   s   create a new empty Sun disklabel
   t   change a partition's system id
   u   change display/entry units
   v   verify the partition table
   w   write table to disk and exit
   x   extra functionality (experts only)
```

You can then print the current partition table typing p :

```
Command (m for help): p

Disk /dev/sdb: 2002 MB, 2002780160 bytes
16 heads, 32 sectors/track, 7640 cylinders
Units = cylinders of 512 * 512 = 262144 bytes
Disk identifier: 0x00000000

   Device Boot   Start    End    Blocks   Id  System
/dev/sdb1            1     17      4336   83  Linux
/dev/sdb2           18   7640   1951488   83  Linux
```

`sdb1` and `sdb2` are two primary partitions. Let's delete both of them:

```
Command (m for help): d
Partition number (1-4): 1

Command (m for help): d
Selected partition 2
```

Now we create a primary partition to hold the root filesystem. Its size can be as small as 1MB for our use :

```
Command (m for help): n
Command action
   e   extended
   p   primary partition (1-4)
p
Partition number (1-4): 1
First cylinder (1-7640, default 1): 1
Last cylinder, +cylinders or +size{K,M,G} (1-7640, default
7640): +1M
```

Then we create another primary partition to hold the general purpose filesystem storing user and application data. Its size can fill the remaining space on the USB disk (accept the last cylinder proposed by default):

```
Command (m for help): n
Command action
   e   extended
   p   primary partition (1-4)
p
Partition number (1-4): 2
First cylinder (6-7640, default 6): 6
Last cylinder, +cylinders or +size{K,M,G} (6-7640, default
7640): 7640
```

If we display the newly created partition table we have:

```
Command (m for help): p

Disk /dev/sdb: 2002 MB, 2002780160 bytes
16 heads, 32 sectors/track, 7640 cylinders
Units = cylinders of 512 * 512 = 262144 bytes
Disk identifier: 0x00000000

   Device Boot   Start    End    Blocks   Id  System
/dev/sdb1            1      5      1264   83  Linux
/dev/sdb2            6   7640   1954560   83  Linux
```

As the partitions are aligned with cylinders, we can notice that the first partition which should be 1MB large is in fact 5 cylinders large, which represents 1.25MB. We could have tuned this according to the actual size of the filesystem if we were short in space.

All we need to do now is to write the partition table on the USB disk. It is written on the first block, also called the MBR (Master Boot Record) :

```
Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.

WARNING: Re-reading the partition table failed with error
16: Device or resource busy.
The kernel still uses the old table.
The new table will be used at the next reboot.
Syncing disks.
```

To reload the new partition table, remove the USB disk and plug it in again.

Data partition on the USB disk

Using the `mkfs.ext3` create a journaled file system on the second partition of the USB disk. Move the contents of the `www/upload/files` directory (in your target root filesystem) into this new partition.

Connect the USB disk to your board while it's running Linux. Using the `dmesg` command, or having a look at the console, see how the kernel detects the partitions and which device names it gives to them.

Modify the setup scripts in your root filesystem to mount the second USB disk partition on `/www/upload/files`.

Reboot your target system and in U-boot, add the below command line argument for the Linux kernel:

```
usb-storage.delay_use=0
```

Now, boot the Linux kernel and with the `mount` command, check that `/www/upload/file` is now a mount point for the second USB disk partition. Also make sure that you can still upload new images, and that these images are listed in the web interface.

By default, Linux waits 5 seconds before initializing USB storage devices. This can be useful for rotating disks which may need time to spin up, but not with all flash devices. If you don't set this, your mount command will probably be issued before the USB disk is ready. Of course, if have a slow USB disk, you may need to keep a non null value.

Adding a tmpfs partition for log files

For the moment, the upload script was storing its log file in `/www/upload/files/upload.log`. To avoid seeing this log file in the directory containing uploaded files, let's store it in `/var/log` instead.

Add the `/var/log/` directory to your root filesystem and modify the startup scripts to mount a tmpfs filesystem on this directory.

Modify the `www/cgi-bin/upload.cfg` configuration file to store the log file in `/var/log/upload.log`. You will loose your log file each time you reboot your system, but that's OK in our system. That's what tmpfs is for: temporary data that you don't need to keep across system reboots.

Reboot your system and check that it works as expected.

Before changing your startup scripts, you may also try your mount command in the running system, to make sure that it works as expected.

Making a SquashFS image

We are going to store the root filesystem in a SquashFS filesystem in the first partition of the USB disk.

In order to create SquashFS images on your host, you need to install the `squashfs-tools` package. Now create a SquashFS image of your NFS root directory.

Caution: read this carefully before proceeding. You could destroy existing partitions on your PC!

Do not make the confusion between the device that is used by your board to represent your USB-disk (probably `/dev/sda1`), and the device that your workstation uses (probably `/dev/sdb1`).

So, don't use the `/dev/sda1` device to reflash your USB drive from your workstation. People have already destroyed their main Windows or Linux partition by making this mistake.

Finally, using the `dd` command, copy the file system image to the first partition of the USB disk.

Once this is done, run the `sync` command to make sure that the writes are flushed to the storage device.

Booting on the SquashFS partition

In the U-boot shell, configure the kernel command line to use the first partition of the USB disk as the root file system. Also add the `rootwait` boot argument, to wait for the USB disk to be fully ready before trying to mount the root filesystem.

Check that your system still works. Congratulations if it does!

If you don't do this, you may get a kernel panic if the disk is not fully ready yet before the kernel tries to mount the root filesystem.

Filesystems – Flash file systems

Objective: Understand flash file systems usage and their integration on the target.

After this lab, you will be able to

Prepare filesystem images and flash them.

Define partitions in embedded flash storage.

Setup

Stay in `/home/<user>/felabs/sysdev/fs`.

Goals

Instead of using an external USB disk as in the previous lab, we will make our system use its internal flash storage.

The root filesystem will still be in a SquashFS filesystem, but this time put on an MTD partition. Read/write data will be stored in a JFFS2 filesystem in another MTD partition.

Here are what the flash partitions will be:

- First partition: kernel (2 MiB)
- Second partition: SquashFS root filesystem (1 MiB)
- Third partition: jffs2 data filesystem (filling all space left)

Filesystem image preparation

Prepare a JFFS2 filesystem image from the `/www/uploads/files` directory from the previous lab, specifying an erase block size of 128KiB. Write down the size of the image file, and check that it is a multiple of the erase block size.

Modify the root filesystem to mount a JFFS2 filesystem on the third flash partition, instead of an ext3 filesystem on the second USB disk partition. Update your SquashFS image.

Enabling the filesystems

Recompile your kernel with support for `JFFS2` and for support for MTD partitions specified in the kernel command line (`CONFIG_MTD_CMDLINE_PARTS`).

Update your kernel image on flash.

MTD partitioning and flashing

Memory layout and partitioning can be defined inside kernel sources, naturally in the `arch/<arch>/<march>/<board>.c` since it is board dependent. Nevertheless, we prefer to define flash partitions at boot time, on the kernel command line, so that these partition perfectly match our needs.

Check the size of the kernel image and the SquashFS image of the root filesystem.

Enter the U-Boot shell and erase NAND flash, from the first MiB boundary following the kernel location, up to the end of the NAND flash (leave the size argument of `nand write` undefined).

Using the `tftp` command, download and flash the SquashFS image at the beginning of the erased flash area.

Using the `tftp` command, download and flash the JFFS2 image at the first MiB boundary following the end of the SquashFS image in flash.

Look at the way default MTD partitions are defined in the kernel sources (`arch/arm/mach-at91/board-usb-a9263.c`)

Set the `bootargs` variable so that you define 3 MTD partitions

- 2 MB for the kernel
- 1 MB for the root filesystem
- The remaining space for data

Don't forget to make the `root` parameter point to the root filesystem in flash.

Boot the target, and by looking at `/proc/mtd`, check that MTD partitions are well configured, and that your system still works as expected.