Forensic study of YAFFS2 filesystem

(Mounting Test Environment)

redactor : buhtig@hashment.com

Table of Content

Chapter 1 : Setting Up the Work Environment

YAFFS2 is a fairly old system (dating back to >2002).
It is not natively included in Linux distributions: the driver must be installed either as a module or directly into the kernel.
My work PC is running Ubuntu 24.04.2 LTS (kernel 6.8.0-60-generic) and gcc 13.3.0-6.
Here are the different attempts that were made:

1.1 All Failures

1.1.1 Compiling kernel sources 6.8.0-60

The first idea was to download the kernel sources, add the YAFFS2 driver, and compile everything. Unfortunately, this failed due to numerous errors.

1.1.2 Creating an environment with Buildroot

I then turned to compiling the kernel using Buildroot.
Many compilation errors arose related to compatibility with the system's native Python version. An older version of Python was required.
To avoid side effects on the installed system, the decision was made—after several hours—to abandon this approach in favor of a virtualized solution.

1.1.3 VirtualBox with an old version of Debian

I used an old version of Ubuntu (12.04 LTS), kernel version 3.2.0-23.
The installation went smoothly, but the package mirrors were no longer available and difficult to find.
Furthermore, communication with the host system did not seem to work (VirtualBox Extension), leaving the virtual image isolated and unable to easily interact with the outside world (certificate issues).
In short → this approach was also abandoned in favor of QEMU.

1.1.4 Compiling kernel 6.8.0-60 + BusyBox for QEMU

The compilation went well despite a few issues—attention was needed to ensure YAFFS2 was built into the kernel.
However, it was impossible to get YAFFS2 to load once the VM was launched, even after adding kernel logs (`dmesg`) into the YAFFS source code before compilation.
Research showed that:

> *The YAFFS2 file system is historically compatible with Linux kernels 2.6.x to 4.x, but since the major changes in the VFS from Linux 5.x onward, many structures, fields, and functions have been removed or modified, making compilation more complex without specific updates.*

The decision was made to switch to a much older kernel that is fully compatible with YAFFS2: kernel 3.2.0 with BusyBox 1.36-1.

1.2 Compiling kernel 3.2.0 + BusyBox 1.36-1

This time, gcc 13.3.0-6 was not compatible with the old 3.2.0 kernel.
I had to install the oldest available Ubuntu package of gcc: version 10.
But the compilation of the old kernel still failed.

In the end, after much research and various more or less successful attempts, the following steps were required:

- Compile and install the gcc-8.5 sources within the current Ubuntu environment
- From gcc 8.5, compile and install the gcc-4.9 sources
- From gcc-4.9, compile the environment that allowed me to begin working

To achieve this, it was ideal to enable compilation using gcc-4.9 throughout the rest of the user session.

One method is to prioritize the directory containing the gcc-4.9 binaries over all others:

```
export PATH=$HOME/gcc-4.9/bin:$PATH
```

To complete my work environment, I need to create 3 files:

- The compiled 3.2.0 kernel (→ file: `bzImage`)
- The initramfs containing all the BusyBox 1.36-1 utilities + some additional tools + an init script launching a shell (→ file: `initramfs.cpio.bz`)
- An ext2 format image file (→ file: `rootfs_container.img`) intended to store the snapshots of the YAFFS2 partition being studied

1.2.1 Compiling kernel 3.2.0

The kernel was difficult to compile, because I need a lot of specificity and I discover them gradually.

- VIRTIO drivers

```
Device Drivers --->
  <*> Virtio drivers --->
    <*> PCI driver for virtio devices
    <*> Virtio block driver
```

- MTD drivers

```
Device                                          Drivers                                         --->
        <*> Memory Technology Device (MTD) support  --->
```

This menu is so big that I decided to activate everything and compile as much as possible in the kernel.

- YAFFS2 driver

```
File Systems - Miscellaneous filesystems
<*>   yaffs2 file system support
-*-     512 byte / page devices
[ ]       Use older-style on-NAND data format with pageStatus byte
[ ]         Lets yaffs do its own ECC
-*-     2048 byte (or larger) / page devices
[*]       Autoselect yaffs2 format
[ ]       Disable yaffs from doing ECC on tags by default
[ ]     Force chunk erase check
[ ]     Empty lost and found on boot
[ ]     Disable yaffs2 block refreshing
[ ]     Disable yaffs2 background processing
[ ]     Disable yaffs2 bad block marking
[*]     Enable yaffs2 xattr support
```

- the EXT2 filesystem

```
File Systems
<*> Second extended fs support
[*]   Ext2 extended attributes
[*]     Ext2 POSIX Access Control Lists
[*]     Ext2 Security Labels
[*]   Ext2 execute in place support
```

Here are the most important elements that must finally be in the `.config` file, before compilation:

```
CONFIG_DEVTMPFS=y                For /dev and /dev/console
CONFIG_DEVTMPFS_MOUNT=y
--------------------------------------------------------------------------------
CONFIG_EXT2_FS_POSIX_ACL=y       For having ext2
CONFIG_EXT2_FS_SECURITY=y
CONFIG_EXT2_FS_XATTR=y
```

```
CONFIG_EXT2_FS_XIP=y
CONFIG_EXT2_FS=y
-----------------------------------------------------------------------------
CONFIG_MTD_NAND_ECC_BCH=y         For having NAND management
CONFIG_MTD_NAND_ECC_SMC=y
CONFIG_MTD_NAND_ECC=y
CONFIG_MTD_NAND_NANDSIM=y
CONFIG_MTD_NAND_PLATFORM=y
CONFIG_MTD_NAND_VERIFY_WRITE=y
CONFIG_MTD_NAND=y
CONFIG_MTDRAM_ABS_POS=0
CONFIG_MTDRAM_ERASE_SIZE=128
CONFIG_MTDRAM_TOTAL_SIZE=4096
CONFIG_MTD_RAM=y
-----------------------------------------------------------------------------
CONFIG_VIRTIO_BALLOON=y           For virtio devices
CONFIG_VIRTIO_BLK=y
CONFIG_VIRTIO_MMIO=y
CONFIG_VIRTIO_PCI=y
CONFIG_VIRTIO_RING=y
CONFIG_VIRTIO=y
-----------------------------------------------------------------------------
CONFIG_YAFFS_AUTO_YAFFS2=y        For YAFFS2
CONFIG_YAFFS_FS=y
CONFIG_YAFFS_XATTR=y
CONFIG_YAFFS_YAFFS1=y
CONFIG_YAFFS_YAFFS2=y
```

Once the compilation finish, I have a file named `arch/x86_64/boot/bzImage` : this is the kernel.

1.2.2 busybox-1.36-1 compilation

> *The compilation MUST be **static** : it's very, very, very important as I don't have modules.*

```
#> cd busybox
#busybox> make menuconfig
```

Then I've selected theses options :

```
Settings
--- Build Options
[*] Build static binary (no shared libs) ← Very Very important to have image boot
```

An error occurred about the utility named « tc » → desactivated because not used.

```
Networking Utilities
[*] Enable Unix domain socket support (usually not needed)
[snip]
[*] netcat (11 kb)
[*]   Netcat server options (-l)
[*]   Netcat extensions (-eiw and -f FILE)
[*]   Netcat 1.10 compatibility (+2.5k)
[snip]
[ ] tc (8.3 kb)
```

After having select all I want, I check the `.config` file before compilation, I've :

```
CONFIG_STATIC=y
CONFIG_STATIC_LIBGCC=y
CONFIG_FEATURE_UNIX_LOCAL=y
CONFIG_NETCAT=y
```

The static ompilation is made with :

```
#> make STATIC=y LDFLAGS=-static
```

Once the compilation finished, I need to create the initramfs.cpio.gz file which is required to boot the QEMU image :

```
user@.> cd busybox
user@busybox> make menuconfig
user@busybox> make STATIC=y LDFLAGS=-static
user@busybox> mkdir ../initramfs
user@busybox> make PREFIX=../initramfs install
        -- create init file --
user@busybox> echo '#!/bin/sh'            > init
user@busybox> echo 'exec > /dev/console 2>&1'     >> init
user@busybox> echo 'exec /bin/sh'         >> init
        -- put owner/permissions --
user@busybox> chown -R root:root *
user@busybox> chmod +x init
        -- creat initramfs file USING root--
user@busybox> sudo su -
root@busybox> find . -print0 | cpio --null -ov --format=newc | gzip -9 > ../initramfs.cpio.gz
```

However, after using BusyBox, it turned out that the nandwrite utility included in BusyBox does not support the -o option, which is essential for my use. This option allows writing the Out Of Band (OOB) areas to the simulated NAND.

→ Therefore, I will need to replace the MTD utilities by **statically** compiling their source code.

The nc and netcat utilities do not support the -U option, which is used to create a UNIX socket.

→ I will need to integrate another utility that allows this: I chose the socat utility, which I will compile **statically**.

Once these utilities are compiled, I need to copy them into the /opt/bin directory of BusyBox. Indeed, nandwrite and nanddump are located in /usr/sbin, so placing the newly compiled utilities in initramfs/opt/bin and giving this directory priority in the PATH will allow them to be used instead (→ done in the /init script).

I also remembered to rebuild the initramfs.cpio.gz file by following the procedure described above.

1.2.3 The ext2 Image

Its purpose is to store the various snapshots taken during the analysis of the YAFFS2 partition.

Its size must be large enough since the YAFFS2 image will be 64MB.

I choose to work only with compressed snapshots, so 20MB will be enough.

To do this, a directory must be created and populated with the desired contents:

At a minimum, the blank YAFFS2 image must be placed there.

```
user@.> mkdir tmp_img
user@.> cp snapshot_00_empty.bin  tmp_img/
user@.> gzip -9 tmp_img/snapshot_00_empty.bin
        -- Generate ext2 image --
user@.> genext2fs -b 20480 -d tmp_img rootfs_container.img
```

The idea is to mount this image inside the virtual image and store YAFFS2 snapshots in it as the file system is modified.

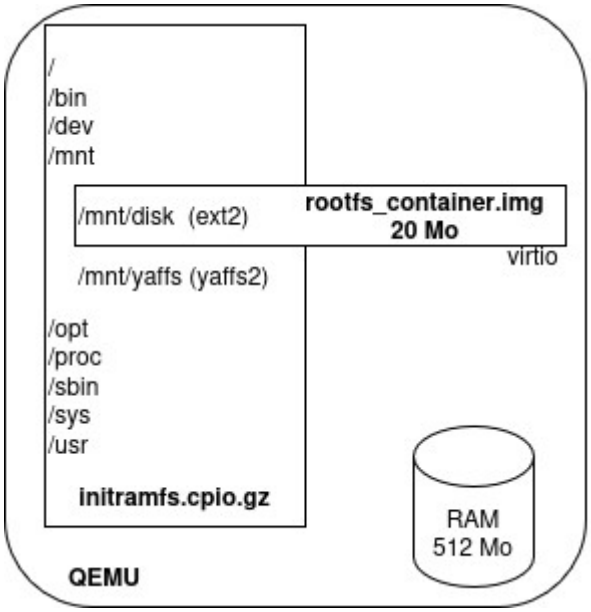
Then, once the virtual machine is shut down, on the host system (Ubuntu 24.04 LTS), I can mount this disk locally to retrieve the files and work on them using the following command:

```
user@.> mkdir recup
user@.> sudo mount -t ext2 -o loop  rootfs_container.img  recup/
```

Indeed, there is nothing within the virtual image to analyze the contents.

1.2.4 Final Virtual Environment

Here is the detailed environment for working :



This environment allows booting a minimal system (without modules) thanks to the drivers compiled directly into the kernel. When modules are included, the initramfs.cpio.gz file becomes too large, and the 3.2.0 kernel refuses to boot.
This environment includes:

1.2.4.1 An initramfs disk

It contains BusyBox as well as all the necessary utilities: mtd-utils (nandwrite, nanddump), netcat, and socat.

1.2.4.2 An ext2-format disk

It allows mounting an ext2 partition using the virtio driver.
I added special files to make future simulations, it contains:

```
$ ls -lR
.:
total 76
drwxrwxr-x 3   4096 juin  24 10:43 files_in_fs
drwxrwxr-x 2 user user  4096 mai   19 16:56 result
-rw-r--r-- 1 user user 67217 mai   22 11:18 snapshot_00_empty.bin.gz

./files_in_fs:
total 28
-rw-rw-r-- 1 user user 6639 juin  24 10:41 big_lorem.txt
-rw-rw-r-- 1 user user   56 juin  24 10:43 big_lorem.txt.sha1
drwxrwxr-x 2 user user 4096 mai   19 16:54 directory
-rw-rw-r-- 1 user user   45 mai   19 16:54 pangram.txt
-rw-rw-r-- 1 user user    5 mai   19 16:54 test1.txt
-rw-rw-r-- 1 user user    5 mai   19 16:54 test2.txt

./files_in_fs/directory:
total 4
-rw-rw-r-- 1 user user 445 mai   19 16:54 lorem.txt
```

```
./result:
total 0
```

The usefulness of this partition lies in the fact that it:

- Provides access to base files (notably the “blank” image to flash into the NAND).
- Stores work-related data generated in this environment, such as disk snapshots.

1.2.5 Launching QEMU

The virtual image will be launched using the following command:

```
#> qemu-system-x86_64  \
        -kernel ./bzImage \
       -m 512M \
        -initrd ./initramfs.cpio.gz \
       -drive file=./rootfs_container.img,format=raw,if=virtio \
       -d guest_errors \
       -nographic \
        -serial mon:stdio \
        -append "console=ttyS0 console=/dev/console init=/init"
```

We can clearly see the three files generated earlier.