Flash Devices

Zilogic Systems

1. Why MTD?

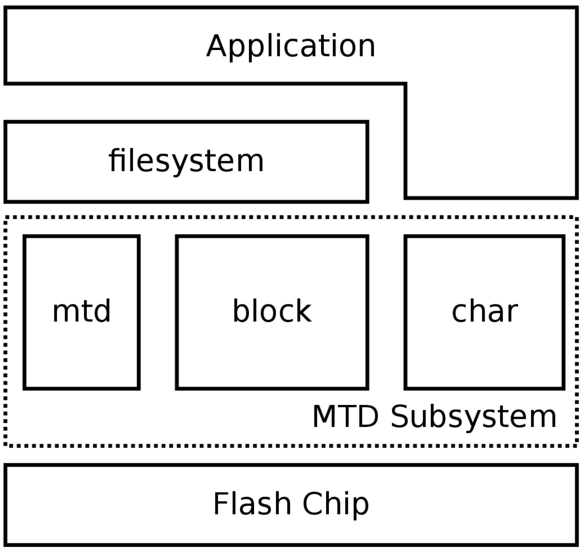
- All hardware devices are abstracted as files in Linux.
- Flash devices should also be abstracted as files. They should be accessible to the user as either character device or a block device.
- Since we would like to use Flash for file storage, block devices seem to be a natural choice. This will allow us to create a filesystem in the Flash.
- Block devices support a read operation and write operation. In the case of a Flash device, in the write operation, the block to be modified has to be read to memory, modified in memory, the block in Flash has to be erased and and the modified data has to be written back to block.
- There are two problems when Flash device is used this way.
- First, during the read-modify-erase-write operation, if the power goes after erase, data worth of 128KB can be lost. Since the block size of Flashes are generally in the order of 128KB.
- Second, if the Flash is treated as regular block device, and regular filesystem is used on top of the block device. All modification to a file will be done to the same block. This will cause some portions of the Flash to get worn out more quickly than the other portions of the Flash, leading to shorter lifespan for the Flash device.
- A special device type for Flash devices was created — MTD (Memory Technology Device), these provide a separate erase operation apart from the regular read / write operations. Also filesystems dedicated to Flash that can work on top of the MTD devices are available.

2. Block Device vs. MTD

Operation Unit	Block device operates on **sectors** of size 512 or 1024 bytes. MTD operates on **erase blocks** of size ~128KB.
Operations	Block device supports 2 main operations: read sector and write sector. MTD supports 3 main operations: read from erase-block, write to erase-block, and erase erase-block
Bad Memory	In modern hard drives, bad sectors are re-mapped and hidden by hardware. In Flash devices bad erase blocks are not hidden and should be dealt with, in software.

3. Using MTDs

Figure 1. Char and block device emulation



- A char device and block device layers are also provided on top of the Flash device.
- From an application point of view the Flash device can be accessed in 3 different ways.

FS on MTD	An MTD aware filesystem can be used to store files in Flash. Example: JFFS2
FS on Block Device	An MTD un-aware filesystem can be used on top of the emulated block device. This can cause Flash to wear out. But is useful if the filesystem is read-only. Example: ext2 on Flash, squashfs on Flash.
Raw Access	The bytes in the Flash device can be accessed as if it were contents of a file, through the MTD char device interface. Special ioctls are provided for Flash specific operations like erase.

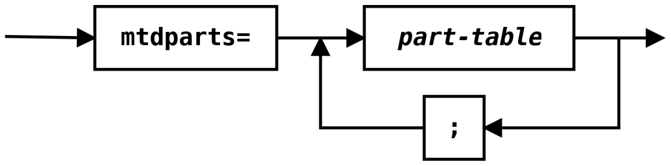
- To access MTD as char/block devices, the following device nodes can be used.

Device Name	Major	Minor	Description
`/dev/mtdN`	90	2*N	Character device
`/dev/mtdrN`	90	2*N+1	Read-only Char. device
`/dev/mtdblockN`	31	N	Block device

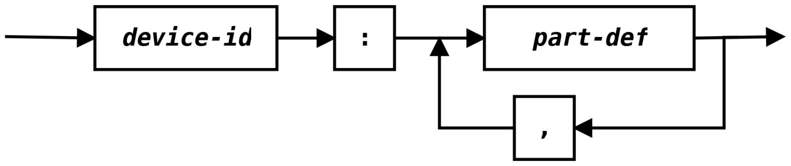
4. MTD Partitions

- Just as with a hard-disk, MTDs can also be partitioned.
- But unlike a hard-disk, the partition table is specified as a kernel boot argument.
- Partition table for multiple devices are specified in a single argument `mtdparts`.
- The general format is given below.

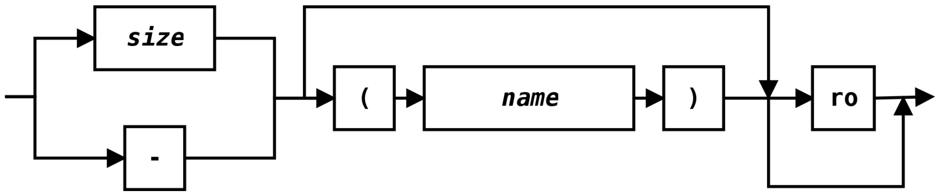
Figure 2. `mtdparts` format



part-table:



part-def:



- The `device-id` is the name of the MTD to be partitioned. The name can be obtained from `/proc/mtd`.
- Examples:

```
mtdparts=nor:512k(u-boot),2M(kernel),-(ramdisk)
mtdparts=nor:512k(u-boot),2M(kernel);nand:128M(db),-(fs)
```

- When a Flash device is partition, the device file of the Flash device itself is no longer available. The device node for the partitions, appear as if they were separate Flash devices.
- The Flash partitions and the corresponding MTD device name is available in `/proc/mtd`.

```
$ cat /proc/mtd
dev:    size   erasesize  name
mtd0: 00004200 00000210 "bootstrap"
mtd1: 00004200 00000210 "env"
mtd2: 00039c00 00000210 "u-boot"
mtd3: 001ce000 00000210 "kernel"
mtd4: 00210000 00000210 "fs"
```

5. Using MTDs, Revisited - NOR Flash

- Loading the kernel or an image on to an MTD partition.

```
$ tftp -g -r uImage my.tftp.server
# flashcp -v uImage /dev/mtd1
```

- To work with MTD character devices, utilities are available from the MTD website.
- Creating an empty flash device and mounting it with JFFS2 as file system.

```
# flash_eraseall /dev/mtd4
# mount -t jffs2 /dev/mtdblock4 /mnt/df
```

- Creating a JFFS2 filesystem image, in the host.

```
host$ mkfs.jffs2 -r /home/resources/tools/ramdisk -n -e 0x2100 -o root.jffs2
target# flash_eraseall /dev/mtd4
target# flashcp -v root.jffs2 /dev/mtd4
target# mount -t jffs2 /dev/mtdblock4 /mnt/df
```

- `-r` specifies the path of the root file system.
- `-e` specifies the erase block size.
- `-o` specifies the image file to be created.
- `-n` specifies not to write clean markers to the beginning of each erase block.
- To boot the kernel with jffs2 as root file system, the following kernel boot arguments should be provided.

```
rootfstype=jffs2 root=/dev/mtdblock4
```

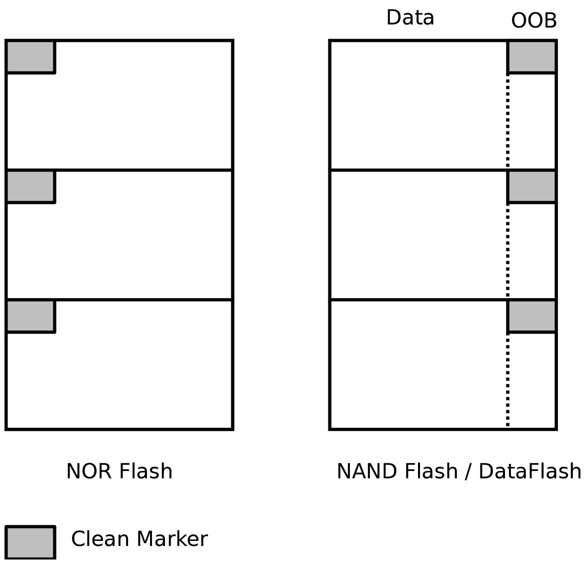
6. Eraseblock Size

- The DataFlash allows the data to be erased in pages or blocks.
- The smallest erasable unit on a DataFlash is a page. The page size of the DataFlash present on the Olimex board is `0x210`. This is indicated in the contents of `/proc/mtd`.
- But JFFS2 was designed for devices with large erase sizes. JFFS2 expects an erase size of atleast 8KB.
- A solution to this problem is to group together adjacent pages in the DataFlash and treat it as one big erase block. This is done by specifying a larger erase size which is a multiple of the smallest erase size.
- `0x2100` is a suitable erase block size, since it is a multiple of `0x210` and is larger than the min. 8KB requirement of JFFS2.

7. Cleanmarker Nodes

- If the power goes off when a flash block is being erased, the block is not completely erased.
- The block reads all 0xFFs, but there might be unstable bits that read as 0 sometimes, and 1 some other times.
- So it is not possible to identify if a block is erased by just looking if all bytes are 0xFF.
- To overcome this, JFFS2 always writes a cleanmarker node, once it is erased. If a cleanmarker node is present then the block has been properly erased.
- In NOR flashes, the cleanmarker node is written at the beginning of the block.
- In NAND flash and DataFlash, each page has an Out-Of-Band (OOB) area which is used for storing meta-data like, bad-block marker, Error correction codes, etc. In the NAND flash and DataFlash, the cleanmarker is written to the OOB area of the first page.

Figure 3. Cleanmarker Nodes



- When JFFS2 images are created for NOR flash, the cleanmarker is included as part of the image.
- When JFFS2 images are created for NAND flash and DataFlash, the image does not contain the OOB data. And hence cleanmarkers should no be part of the image. This is indicated to the `mkfs.jffs2` utility using the `-n` option.

8. Using MTDs - NAND Flash

- Creating JFFS2 filesystem for NAND flash.

```
host$ mkfs.jffs2 -r /home/resources/tools/ramdisk -n -e 0x20000 -o root.jffs2
```

- Erasing MTD partition for NAND Flash .

```
target# flash_eraseall -j /dev/mtd0
```

- `-j` will inserts the cleanmarker at the beginning of each block, so that the JFFS2 won't redo the erase operation.

```
target# nandwrite -p /dev/mtd0 root.jffs2
```

- `-p` will pad to page size.

References

- Information about MTD subsystem is available from the MTD FAQ http://www.linux-mtd.infradead.org/doc/general.html
- Information about JFFS2 filesystems is available from the JFFS2 FAQ http://www.linux-mtd.infradead.org/faq/jffs2.html
- MTD device node info is available from Documentation/devices.txt in the kernel source tree.
- `mtdparts` boot argument format is described in drivers/mtd/cmdlinepart.c in the kernel source tree.