UBI File System

Introduction

UBIFS file-system stands for UBI File System. UBI stands for "Unsorted Block Images". UBIFS is a flash file system, which means it is designed to work with flash devices. It is important to understand, that UBIFS is completely different to any traditional file-system in Linux, like Ext2, XFS, JFS, etc. UBIFS represents a separate class of file-systems which work with MTD devices, not block devices. The other Linux file-system of this class is JFFS2.

To make it more clear, here is a small comparison of MTD devices and block devices.

1 MTD devices represent flash devices and they consist of eraseblocks of
  rather large size, typically about 128KiB. Block devices consist of small blocks, typically 512 bytes.

2 MTD devices support 3 main operations - read from some offset within an
  eraseblock, write to some offset within an eraseblock, and erase a whole eraseblock. Block devices support 2 main operations - read a whole block and write a whole block.

3 The whole eraseblock has to be erased before it becomes possible to
  re-write its contents. Blocks may be just re-written.

4 Eraseblocks become worn out after some number of erase cycles -
  typically 100K-1G for SLC NAND and NOR flashes, and 1K-10K for MLC NAND flashes. Blocks do not have the wear-out property.

5 Eraseblocks may become bad (only on NAND flashes) and software should
  deal with this. Blocks on hard drives typically do not become bad, because hardware has mechanisms to substitute bad blocks, at least in modern LBA disks.

It should be quite obvious why UBIFS is very different to traditional file-systems.

UBIFS works on top of UBI. UBI is a separate software layer which may be found in drivers/mtd/ubi. UBI is basically a volume management and wear-leveling layer. It provides so called UBI volumes which is a higher level abstraction than a MTD device. The programming model of UBI devices is very similar to MTD devices - they still consist of large eraseblocks, they have read/write/erase operations, but UBI devices are devoid of limitations like wear and bad blocks (items 4 and 5 in the above list).

In a sense, UBIFS is a next generation of JFFS2 file-system, but it is very different and incompatible to JFFS2. The following are the main differences.

- JFFS2 works on top of MTD devices, UBIFS depends on UBI and works on top of UBI volumes.
- JFFS2 does not have on-media index and has to build it while mounting, which requires full media scan. UBIFS maintains the FS indexing information on the flash media and does not require full media scan, so it mounts many times faster than JFFS2.
- JFFS2 is a write-through file-system, while UBIFS supports write-back, which makes UBIFS much faster on writes.

Similarly to JFFS2, UBIFS supports on-the-flight compression which makes it possible to fit quite a lot of data to the flash.

Similarly to JFFS2, UBIFS is tolerant of unclean reboots and power-cuts. It does not need stuff like fsck.ext2. UBIFS automatically replays its journal and recovers from crashes, ensuring that the on-flash data structures are consistent.

UBIFS scales logarithmically (most of the data structures it uses are trees), so the mount time and memory consumption do not linearly depend on the flash size, like in case of JFFS2. This is because UBIFS maintains the FS index on the flash media. However, UBIFS depends on UBI, which scales linearly. So overall UBI/UBIFS stack scales linearly. Nevertheless, UBI/UBIFS scales considerably better than JFFS2.

The authors of UBIFS believe, that it is possible to develop UBI2 which would scale logarithmically as well. UBI2 would support the same API as UBI, but it would be binary incompatible to UBI. So UBIFS would not need to be changed to use UBI2

Mount options

(*) == default.

bulk_read	read more in one go to take advantage of flash media that read faster sequentially
no_bulk_read (*)	do not bulk-read
no_chk_data_crc (*)	skip checking of CRCs on data nodes in order to improve read performance. Use this option only if the flash media is highly reliable. The effect of this option is that corruption of the contents of a file can go unnoticed.
chk_data_crc	do not skip checking CRCs on data nodes
compr=none	override default compressor and set it to "none"
compr=lzo	override default compressor and set it to "lzo"
compr=zlib	override default compressor and set it to "zlib"
auth_key=	specify the key used for authenticating the filesystem. Passing this option makes authentication mandatory. The passed key must be present in the kernel keyring and must be of type 'logon'

auth_hash_name=	The hash algorithm used for authentication. Used for both hashing and for creating HMACs. Typical values include "sha256" or "sha512"

Quick usage instructions

The UBI volume to mount is specified using "ubiX_Y" or "ubiX:NAME" syntax, where "X" is UBI device number, "Y" is UBI volume number, and "NAME" is UBI volume name.

Mount volume 0 on UBI device 0 to /mnt/ubifs:

```
$ mount -t ubifs ubi0_0 /mnt/ubifs
```

Mount "rootfs" volume of UBI device 0 to /mnt/ubifs ("rootfs" is volume name):

```
$ mount -t ubifs ubi0:rootfs /mnt/ubifs
```

The following is an example of the kernel boot arguments to attach mtd0 to UBI and mount volume "rootfs": ubi.mtd=0 root=ubi0:rootfs rootfstype=ubifs

References

UBIFS documentation and FAQ/HOWTO at the MTD web site:

- http://www.linux-mtd.infradead.org/doc/ubifs.html
- http://www.linux-mtd.infradead.org/faq/ubifs.html