The QNX6 Filesystem

The qnx6fs is used by newer QNX operating system versions. (e.g. Neutrino) It got introduced in QNX 6.4.0 and is used default since 6.4.1.

Option

mmi_fs Mount filesystem as used for example by Audi MMI 3G system

Specification

qnx6fs shares many properties with traditional Unix filesystems. It has the concepts of blocks, inodes and directories.

On QNX it is possible to create little endian and big endian qnx6 filesystems. This feature makes it possible to create and use a different endianness fs for the target (QNX is used on quite a range of embedded systems) platform running on a different endianness.

The Linux driver handles endianness transparently. (LE and BE)

Blocks

The space in the device or file is split up into blocks. These are a fixed size of 512, 1024, 2048 or 4096, which is decided when the filesystem is created.

Blockpointers are 32bit, so the maximum space that can be addressed is 2^32 * 4096 bytes or 16TB

The superblocks

The superblock contains all global information about the filesystem. Each qnx6fs got two superblocks, each one having a 64bit serial number. That serial number is used to identify the "active" superblock. In write mode with reach new snapshot (after each synchronous write), the serial of the new master superblock is increased (old superblock serial + 1)

So basically the snapshot functionality is realized by an atomic final update of the serial number. Before updating that serial, all modifications are done by copying all modified blocks during that specific write request (or period) and building up a new (stable) filesystem structure under the inactive superblock.

Each superblock holds a set of root inodes for the different filesystem parts. (Inode, Bitmap and Longfilenames) Each of these root nodes holds information like total size of the stored data and the addressing levels in that specific tree. If the level value is 0, up to 16 direct blocks can be addressed by each node.

Level 1 adds an additional indirect addressing level where each indirect addressing block holds up to blocksize / 4 bytes pointers to data blocks. Level 2 adds an additional indirect addressing block level (so, already up to 16 * 256 * 256 = 1048576 blocks that can be addressed by such a tree).

Unused block pointers are always set to ~0 - regardless of root node, indirect addressing blocks or inodes.

Data leaves are always on the lowest level. So no data is stored on upper tree levels.

The first Superblock is located at 0x2000. (0x2000 is the bootblock size) The Audi MMI 3G first superblock directly starts at byte 0.

Second superblock position can either be calculated from the superblock information (total number of filesystem blocks) or by taking the highest device address, zeroing the last 3 bytes and then subtracting 0x1000 from that address.

0x1000 is the size reserved for each superblock - regardless of the blocksize of the filesystem.

Inodes

Each object in the filesystem is represented by an inode. (index node) The inode structure contains pointers to the filesystem blocks which contain the data held in the object and all of the metadata about an object except its longname. (filenames longer than 27 characters) The metadata about an object includes the permissions, owner, group, flags, size, number of blocks used, access time, change time and modification time.

Object mode field is POSIX format. (which makes things easier)

There are also pointers to the first 16 blocks, if the object data can be addressed with 16 direct blocks.

For more than 16 blocks an indirect addressing in form of another tree is used. (scheme is the same as the one used for the superblock root nodes)

The filesize is stored 64bit. Inode counting starts with 1. (while long filename inodes start with 0)

Directories

A directory is a filesystem object and has an inode just like a file. It is a specially formatted file containing records which associate

each name with an inode number.

'.' inode number points to the directory inode

'..' inode number points to the parent directory inode

Eeach filename record additionally got a filename length field.

One special case are long filenames or subdirectory names.

These got set a filename length field of 0xff in the corresponding directory record plus the longfile inode number also stored in that record.

With that longfilename inode number, the longfilename tree can be walked starting with the superblock longfilename root node pointers.

Special files

Symbolic links are also filesystem objects with inodes. They got a specific bit in the inode mode field identifying them as symbolic link.

The directory entry file inode pointer points to the target file inode.

Hard links got an inode, a directory entry, but a specific mode bit set, no block pointers and the directory file record pointing to the target file inode.

Character and block special devices do not exist in QNX as those files are handled by the QNX kernel/drivers and created in /dev independent of the underlaying filesystem.

Long filenames

Long filenames are stored in a separate addressing tree. The staring point is the longfilename root node in the active superblock.

Each data block (tree leaves) holds one long filename. That filename is limited to 510 bytes. The first two starting bytes are used as length field for the actual filename.

If that structure shall fit for all allowed blocksizes, it is clear why there is a limit of 510 bytes for the actual filename stored.

Bitmap

The qnx6fs filesystem allocation bitmap is stored in a tree under bitmap root node in the superblock and each bit in the bitmap represents one filesystem block.

The first block is block 0, which starts 0x1000 after superblock start. So for a normal qnx6fs 0x3000 (bootblock + superblock) is the physical address at which block 0 is located.

Bits at the end of the last bitmap block are set to 1, if the device is smaller than addressing space in the bitmap.

Bitmap system area

The bitmap itself is divided into three parts.

First the system area, that is split into two halves.

Then userspace.

The requirement for a static, fixed preallocated system area comes from how qnx6fs deals with writes.

Each superblock got it's own half of the system area. So superblock #1 always uses blocks from the lower half while superblock #2 just writes to blocks represented by the upper half bitmap system area bits.

Bitmap blocks, Inode blocks and indirect addressing blocks for those two tree structures are treated as system blocks.

The rational behind that is that a write request can work on a new snapshot (system area of the inactive - resp. lower serial numbered superblock) while at the same time there is still a complete stable filesystem structure in the other half of the system area.

When finished with writing (a sync write is completed, the maximum sync leap time or a filesystem sync is requested), serial of the previously inactive superblock atomically is increased and the fs switches over to that - then stable declared - superblock.

For all data outside the system area, blocks are just copied while writing.