

Nonvolatile File System

Structure used in 68K nodes

Mon, Apr 9, 2007

The scheme for managing files in the nonvolatile memory of 68K-based nodes is described here. This note is prompted by the considerable effort to recover some 200 files in the library node0508. A better backup scheme is required in order to forestall a repeat of this effort. (In PowerPC nodes, a different underlying system is used, based upon a file system support that VxWorks supports.) The allocation logic behind the scheme described here is taken from Knuth, Vol. 1, p. 435–440.

Data structures

The structure that houses the file system is really quite simple. There is a 16-byte header at the start of the nonvolatile memory area used for the file system. The rest of the memory is allocated and freed in a way that is analogous to any memory allocation method.

The 16-byte header format is as follows:

<i>Field</i>	<i>Size</i>	<i>Meaning</i>
<code>free</code>	4	Address of first (or only) free block
<code>fill</code>	4	Always 0
<code>nvSize</code>	4	Total size of nonvolatile file system area including header
<code>mxSize</code>	4	Current size of largest free block

Two types of variable-size blocks are in use: allocated blocks and free blocks. Each block includes an 8-byte header. For an allocated block, the header is:

<i>Field</i>	<i>Size</i>	<i>Meaning</i>
<code>aSize</code>	4	Allocated block size including this 8-byte header
<code>aType</code>	4	Block type, always 0x000000F

For a free block, the header is:

<i>Field</i>	<i>Size</i>	<i>Meaning</i>
<code>fNext</code>	4	Address of the next free block, or 0 if none.
<code>fSize</code>	4	Free block size including this 8-byte header

Note that all allocated sizes and free sizes are multiples of 8 bytes. The `fNext` fields comprise a linked list of free blocks, where the list head is the field `free` in the 16-byte header above.

For 68K program files, currently with 8-character names beginning with either `LOOP` or `PAGE`, the entry point is always at the beginning of the code, which is assumed to start with a `LINK` instruction, for which the first 16-bit word is 0x4E56. Code that copies files knows this. Data files have no such restriction. Any file is assumed to be at least 32 bytes in size.

Memory management

The code that supports all this is found in the file `MA11oc`, including the following functions:

<i>Name</i>	<i>Function</i>
<code>MA11oc</code>	Allocate memory in nonvolatile memory, given a 32-bit size.
<code>MLiber</code>	Liberate (free) memory, given address of block to be freed.
<code>InzA11oc</code>	Initialize nonvolatile area as one large free block.
<code>MSqueeze</code>	Compress allocated blocks so that all free space is coalesced into one block.

The `InzAlloc` call is only made at boot time if the 16-byte header is invalid. To force this to happen, first clear the 16-byte header, then boot. The file system area will then be empty. (As indicated below, the `CODES` table file directory should also be cleared to cleanly start over.) When releasing a block, the code notices if the block just freed is located adjacent to an already free block, and if so, it coalesces the two into one larger block. The overall coalescing performed by `MSqueeze` is not done except at boot time, since other system data structures may be affected.

Any file system needs a file directory. For these systems, the nonvolatile system table `CODES` serves that purpose. Each entry is a 32-byte structure with the following format:

<i>Field</i>	<i>Size</i>	<i>Meaning</i>
<code>tName</code>	4	4-character file type code
<code>aType</code>	4	4-character file name
<code>cSize</code>	4	File content size (not including 8-byte header)
<code>fCksm</code>	4	File check sum, a 32-bit sum of unsigned 16-bit words
<code>fileD</code>	4	File download address, immediately following 8-byte header
<code>fileE</code>	4	Program file entry address, after copying to dynamic allocated memory
<code>vDate</code>	6	File version date, in BCD Yr, Mo, Da, Hr, Mn, Sc.
<code>cCntr</code>	2	Call count diagnostic, #times the file has been copied into dynamic ram.

The concatenation of the `tName` and `fName` fields is often referred to as an 8-character filename. Names that are not 8 characters long should be blank-filled, although none yet exist.

Save/Restore

Note that saving a copy of the file system for backup means that both the `CODES` table and the nonvolatile file area must be saved. When performing a restore, both must be restored. The file system has integrity only when considering both structures as a unity. The `fileD` field in the `CODES` table entry refers to an address in the file area. Under a squeeze operation, which is only done when a node boots, `fileD` addresses may be altered as allocated file blocks are moved.

If it is desired to remove such an unknown block and add it to the free block linked list, there is no easy way to do this as yet. The method used (far too many times) is to manually enter enough of a `CODES` table entry to point to the block, then use the download page `PAGEDNLD` to release that block. This is done by entering 0000 over the size shown on a directory listing fragment, then clicking. The block will be added to the free block list, and the manually entered `CODES` entry will be cleared. The manual entry can be, say, a type name of `AAAA`, the `cSize` entered will be 8 less than the number shown by the `PAGENVOL` diagnostic, and the `fileD` field will be 8 more than the address of the block header shown by `PAGENVOL`. To get the field shown, do a directory listing with the filtering of `A`, say. (All this works because there are no valid file names as yet that begin with `A`.) After having to do this a few times, one's nervousness tends to subside. Ideally, of course, this procedure should never have to be done.

File copying

Copying files between nodes is very commonly used when configuring a new front end node. The download page `PAGEDNLD` provides this service. One specifies a source node and filename, plus a destination node. The transfer takes place using the Classic protocol, with a special `listype#` (76) designed for the purpose. The 14-byte ident format consists of a node#, an 8-byte filename, and a 4-byte offset. Special offset values are used to deliver the file size, checksum and version date.

Version date

The version date is meant to tag the version of a file, since it is often the case that files are

updated, so that some means of distinguishing different versions is necessary. For 68K systems, the date is assigned as the time when the TFTP server LOOPTFTP receives a new file and installs it into the nonvolatile file system. (In PowerPC systems, the date is taken from the development system file's last-modified date.) Because of this, we normally download a new program version only once from the development system via the TFTP protocol. To install the program in another node, we use the file copy mechanism via PAGEDNLD. This practice preserves the version date.

User level

When a file is transferred into a node, it replaces any file of the same name that already exists in that node's file system; the previous version is freed and the new one is allocated. In the case that the file is a local application program file, and if that LA is currently active, an automatic and orderly switch to the the new version takes place. A termination call is made to the active LA, the old dynamic memory copy is released, the new version is allocated and copied into dynamic memory for execution and called to initialize itself. In the rare case that this automatic switching is not desired, merely disable the current LA before downloading the new version.

For a page application that is updated, however, no such automatic switching takes place. In the case that the previous version of the PA is active, one must instead recall the page application to activate the new version just downloaded.

Note that only 68K program files can reside in a 68K node file system. Similarly, only PowerPC program files can reside in a PowerPC file system. This is necessary because two versions of the same program often exist using the same filename, since they provide the same functionality. Data files may reside in either type of node.

Diagnostics

A special page application PAGEVERS allows comparing versions of files in a "target" node with a "reference" node. For each target node file that differs from that in the reference node, it indicates which is newer. It also indicates which files in the target node are not in the reference node. It merely compares the CODES table contents of both nodes. Most commonly, the reference node is a library node. Node0508 is used for 68K files, and node0619 is used for PowerPC files.

A diagnostic page application PAGENVOL can be used to examine the health of the file system area. It looks at each block in the area and checks to see whether all allocated blocks are pointed to by CODES table entries. It also checks that all other blocks are part of the free space linked list of blocks. If it sees a block that is "unknown," it shows the 8-byte header field of that block. One could see this effect if the program is run after a CODES table entry is obliterated. It will likely show that the first 8 bytes are a size longword followed by the constant 0x000000F.

Previous notes

These notes provide additional documentation about nonvolatile memory usage:

<i>Title</i>	<i>#pages</i>	<i>Date</i>
<i>Downloading Programs</i>	3	Aug 28, 1990
<i>TFTP Implementation</i>	2	May 17, 1994
<i>Nonvolatile Memory Analysis</i>	2	Oct 14, 1994
<i>Program Copying</i>	1	Sep 14, 1998
<i>LocAppl Flow Diagram</i>	2	Feb 17, 2000
<i>Program Versions</i>	4	Jan 26, 2001
<i>Local Application Parameters</i>	2	Jul 28, 2003