

RDATA Scanning

Efficiency improvement

Thu, Aug 8, 2002

The entries in the Data Access Table (RDATA) table comprise a series of instructions that are interpreted every 15 Hz cycle to determine how the data pool is to be updated. Since this table must be in nonvolatile memory so as to survive system resets and power losses, it means that many nonvolatile accesses are made every cycle. In the case of the PowerPC version of the system, such accesses are relatively slow. This note describes a means of reducing greatly the number of RDATA accesses to nonvolatile memory every cycle.

The central theme of the scheme is to keep a copy of the RDATA table in fast memory that is actually used online. The nonvolatile RDATA is only used when making changes to the table. In order to make such changes effective, the system must monitor such changes. But there is no automatic recognition when the contents of the table changes, because such changes are often made via the generic Memory Dump page application. This means that the system must regularly check to see whether any changes may have been made, and if so, arrange to make a new working copy of the RDATA table for use online.

The time to produce the copy might be considered a problem, but it can be done relatively efficiently, moving 4 bytes at a time. If its length were 128 entries of 16 bytes each, this might require 512 accesses, which can require a half ms or more. If desired, this copy could be scheduled at a time away from the early part of the cycle, so it does not represent a perturbation on the natural consistent data pool update timing.

During processing, some of the RDATA entry types actually modify some of the fields of the entry. This might imply that any monitoring made to determine whether the table has changed could very often determine that something has been changed when it really has not. A method to eliminate this problem is to keep a pristine carbon copy of the RDATA table in addition to the copy that is actually used online. The monitoring logic would then compare the contents of the carbon copy against the nonvolatile RDATA. So, we then have three copies of RDATA existing in memory at any one time. The first copy would be in active use; the second copy would be the reference carbon copy to be periodically compared against the third (nonvolatile) copy to detect modifications made to the nonvolatile RDATA.

The monitoring logic can be quite sparing of nonvolatile accesses, because only a small part of the table need be monitored each cycle. Suppose that 4 entries are monitored each cycle, requiring about 20 microseconds. An entire table of 128 entries can thus be checked in 32 cycles, or about 2 seconds. Discovery of a change could prompt the copy operation right away, with at most a 2 second latency after making a change before the change is effective.

There is a temptation that must be avoided. Thinking that RDATA entries are only changed via slow interactive alterations made with the Memory Dump page, one might think that updates of changes could be made piecemeal, without copying the entire table at one time. This would be a mistake, because changes to large parts of the table, or even all of it, can be made at one time by downloading, especially from a computer running the RDATA Editor program that Bob Florian developed. Once a change has been noticed by the monitoring logic, the entire table should be replicated to produce the other two copies.

This scheme does not remove from the user who modifies an RDATA entry via the Memory Dump page the necessity to exercise due care over the changes made at each step. Even

though the table being modified is the nonvolatile version of the table and not the working copy, any change that is made may cause the online copy to be updated almost immediately. One should always make such modifications carefully, recognizing that the table being modified is effectively “live.” One common approach when editing a specific entry is to disable it, traditionally by setting the sign bit in the type number (first) byte of the entry, then make whatever changes are desired, then re-enable the entry by clearing the sign bit. Most people will likely find that using the RDATA Editor program is a more friendly means of changing RDATA table entries.

When making the copies, it can be recognized what the actual effective length of the table is. One might assume that encountering 4 empty table entries in sequence, say, would be enough to denote the end of the table, storing the real number of entries so that processing the table could stop at that point. In addition, it would be possible to compress the table, removing entries that are disabled, when building the online copy of the table. The carbon copy, however, must not be compressed, in order to facilitate the monitoring logic. This compression possibility may not be worth the effort, or the possible resulting confusion.

Monitoring logic

Check four entries every 15 Hz cycle, following the completion of RDATA processing. Scan through nonvolatile RDATA until finding a set of four entries that are all empty. This determines the working size of the RDATA. (When scanning the working copy, the scan will only go this far.) If a change is detected, build new copies of the table, recording suitable diagnostics about it. Restart the monitoring process at the beginning. Restart the ETRDATA timing table that measures the elapsed time of each RDATA entry’s execution, by writing 0xFFFF in the third word, since the table entries may now have been rearranged.

New variables

<i>Name</i>	<i>Size</i>	<i>Meaning</i>
monX	2	monitoring index
nEnt	2	#entries in allocated RDATA
tCopy	2	Time to complete copy, in μ s
nCopy	2	#times copy has been made
dCopy	8	Date-time of last copy
—	2	spare
nEntW	2	#entries to scan in RDATA table
nonV	4	Ptr to nonvolatile RDATA
carb	4	Ptr to carbon copy of nonvolatile RDATA
work	4	Ptr to working copy of RDATA

Location of variables

A low memory variable should be used for the ptr to the working copy of RDATA and/or it should be located in a fixed area of memory. The working copy can include a header of 32 bytes, say, that can hold the relevant variables. Use 0x698 for the low memory ptr variable. This ptr is initialized by InzRData at reset time. If it is NULL, it means that the new logic is not being used, so there is no copy of RDATA being used, there is no header structure, and the nonvolatile RDATA should be referenced instead. It may be useful to find a means of booting a system in a way that avoids the fast method for timing measurements.

Implementation details

The modules that require changes for this implementation, as done for the MC68040-based IRM code, are as follows:

RdAD.a

The bulk of the changes are in this module, including the new `CpyRData` and `MonRData` routines. Changes in the `InzRData` routine create the initial copy scheme support, including allocating a clearing a carbon copy block, and ending with a call to `CpyRData`, which produces the first copies of RDATA. Changes made to the `RdAD` routine itself allow discovery of the working copy of RDATA, when it exists, and to call `MonRData` after `RData` processing to watch for changes in the nonvolatile RDATA, presumably made via the Memory Dump page or by the PC-based RDATA Editor, which provides a more user-friendly interface for editing the RDATA table entries.

InzSys.a

Remove the call to `InzRData`, since it now allocates memory. `pSOS` has not yet started.

Update.a

Install the call to `InzData` here, during `Update` task initialization.

LTT.a

`Listype 30` now uses a new read-type 29 routine `RdRData`.

ReadType.a

New `RdRData` routine.

ReadTypG.a

Glue for `RdRData` routine.

Memory support

In the IRM system code, the implementation of this feature places the RDATA copy variables structure, sketched above, at `0x1F0FC0`. As mentioned above, a low memory global, located at `0x698`, points to this structure, when the scheme is enabled. Up to 64 bytes can be used for this structure, although only 32 bytes are defined so far. The working copy of RDATA begins at the address `0x1F1000`, bearing a memorable relationship to the `0x401000` base address of RDATA in nonvolatile memory. (Find the RDATA copy variables structure by starting at `0x1F1000` and backing up one Memory Dump page.) The carbon copy of RDATA is allocated from dynamic memory, including a 16-byte header, which has no use beyond the first basic eight bytes, but it is convenient to have a header whose size matches the size of an RDATA entry.

Comments

Using `node0509` as a test vehicle, and comparing the execution time between using the new scheme and not, there is a savings of about $200\ \mu\text{s}$ out of a typical `RData` execution time of $1900\ \mu\text{s}$, which can be explained by the shortening of the RDATA scanning to the number of entries actually in the table, rather than scanning the entire table whether it is needed or not. Again, there was no expectation of performance improvement for the IRM. The data for the PowerPC case should look a lot better. (This ignores the time that Linac nodes must await replies from SRMs, of course.)