

# CODES Table Search

*System optimization*

Tue, May 21, 2002

This note describes a simple scheme that has been implemented to improve the performance the majority of CODES table searches. The idea behind the scheme is to retain knowledge of the result of a search, so that when the same "file name" is sought the next time, the knowledge of where it was found the last time can be used to full advantage.

Searches are made of the CODES table for several different purposes. A very common purpose is to lookup the execution address for a given file so that its code can be invoked. For example, during Data Access Table processing, there is one special DAT entry whose meaning is "execute all the local applications that are enabled." For every enabled LA, a search is made to find its execution address so it can be called. This happens every 15 Hz cycle, of course, so there is potential for many wasted CPU cycles in searching for a match by file name. The number of CODES table entries used in a typical node might be 30, including both page and local application program files. The maximum size is normally 64 entries.

Another reason to do a search of CODES is to find the execution address of a local application that supports a network protocol. A network message comes in, and the system must invoke the appropriate LA to handle it, assuming it is not a protocol handled by the system itself.

A third reason for a search is when the Application task, which manages execution of page applications, must invoke that application. Only one is available at a time, and its entry point is retained anyway, but updating the index page requires a scan of all files in that index.

Each Local Application Table (LATBL) entry includes a 4-character local application file name, to which a 'LOOP' prefix is added to produce the 8-character file name to be found in CODES. Within the 32-byte LATBL entry, a byte was used to hold the execution time for that program (in half ms units) the last time it was initialized. But this had limited value, so the new scheme appropriates this byte to hold the CODES table index (entry number) where any search should begin. The PROGPTR code has been modified so that it accepts a pointer to such a byte variable, and it returns the matching CODES index via this byte variable. The effect is that when a search is made of a given file name, the search begins at the very entry where a match is most likely. It is not important if (during software updates, say) the optimal index should change, because the next time a search for that same file name is made, the index value will be updated, so that any subsequent searches will be optimized. The index is only a hint of where to start; the code will search all entries until a match is found.

For the case of page applications, the Application task now keeps an array of 32 bytes, one for each page, that it uses when it makes calls to the PROGPTR routine that in turn calls a search routine to find a matching CODES table entry for a given page application.

When access is made for program file data, say via Page D, there is no need to keep such records, as any file name at all might appear in the request. Such searches merely start at the beginning and scan until a match is found or until the end is reached.

While it is nice to know that useless searches are not being performed in the system code, it is even better for the case of the PowerPC nodes, because of the slow access to nonvolatile memory that holds most of the tables that need to be searched, including the CODES table.