UCRL--89207

DE84 012590

Improvement in MFTF Data Base System Response Times

Neil C. Lang
Bron C. Nelson

Proceedings of the 10th Symposium
on Fusion Energy
Fusion Energy Conference
Philadelphia, PA
December 5-9, 1983

November 28, 1983

IMPROVEMENTS IN MFTF DATA BASE SYSTEM RESPONSE TIMES

N. C. Lang, R. C. Nelson
Lawrence Livermore National Laboratory
P. O. Box 5511 L-535
Livermore, CA 94550

Abstract

The Supervisory Control and Diagnostic System for the Mirror Fusion Test Facility (MFTF) has been designed as an event driven system. To this end we have designed a data base notification facility in which a task can request that it be loaded and started whenever an element in the data base is changed beyond some user defined range. Our initial implementation of the notify facility exhibited marginal response times whenever a data base table with a large number of outstanding notifies was written into. In this paper we discuss the sources of the slow response and describe in detail a new structure for the list of notifies which minimizes search time resulting in significantly faster response.

"Work performed under the auspices of the U.S. Department of Energy by the Lawrence Livermore National Laboratory under contract number W-7405-ENG-48."

Introduction

The Supervisory Control and Diagnostics System (SCDS), in order to monitor, control, and collect data from the Mirror Fusion Test Facility (MFTF), must be able to recognize and respond to a variety of externally generated events. One class of such events is a change in the value of an element in the data base. We have designed and implemented a data base *notification facility* (described in some detail previously[1]) that permits tasks that are interested in specific elements in the data base to be notified whenever the values of those elements change. Tasks can request to be informed only when the change is larger than some preset value, or upon any updating of that element. This approach has significant advantages over some more traditional monitoring techniques. First the notification facility provides for decoupling between the tasks that write new values into the data base and those that need to *know about* that value for display or control purposes. The value of this independence can be shown in a simple example: consider the activity involved in monitoring a set of gas pressures. The task that actually measures the gas pressure need only write the current value into the data base; all the other tasks that need to know the value of the pressure (e.g. a status display or emergency pump system) simply request that they be *notified* whenever that parameter is updated. None of the tasks need to know about each other. Secondly any task watching a specific element in the data base can avoid constant and costly polling of the data base to see if the element has changed.

Description of System

SCDS implements the notification process by attaching to the data base table containing the parameter to be watched, a record (referred to in SCDS as a "notify") indicating the parameter of interest and a reference to the task making the request. We will again use the example of the gas pressure display to briefly review the user and data base views of the notification process. When an operator requests a gas pressure display, the display module, the task that creates and updates the user display, first starts notifies on the corresponding entries in the appropriate data base table or tables. The data base itself creates these new notifies and adds them to the end of the linked list of notifies attached to the relevant table. Whenever any task updates any part of the table, the linked list of notifies for that table is searched to locate any notifies that are watching the parameter that was updated. If a notify is triggered, a message is sent to the interested task. Since the notification message does not contain the newly written value, the task must then read the data base for the new value. Also in most cases, the notify is temporarily deactivated after a notification is sent and the user task must also reactivate or continue the notify; this again requires searching the notify list. This protocol is necessary to prevent rapid and gratuitous updates which would be barely visible to the operator, and also ensures that the most recent value is in fact displayed. Eventually the operator cancels the display; the display module then issues a stop notify command which causes the data base to permanently deactivate that notify and remove it from the linked list.

Performance Issues

Our initial experience with the notification system indicated that it performed as designed, but that the response time (defined here as the time between the writing of a new value into the data base and the updating of the corresponding display) deteriorated under conditions of heavy data base usage. Since each notify list has to be searched completely (the list is in no particular order other than order of arrival) following every update of the table, and searched once again by every task that has been notified, the length of the list and hence the time required to search that list was a major factor contributing to the observed delays.

Since neither the Perkin-Elmer computers used in SCDS nor their OS32 operating system supports virtual memory, and since the tables are too large to be always memory resident, the data base itself is designed as a software implemented 'virtual memory' system in which data of interest is 'paged' in from the disk when required. A least-recently-used algorithm is used to pick the page currently in memory that will be overwritten with the new data. The data base currently provides for 50 pages each containing 256 bytes (a disk sector). Paging is a time consuming operation; approximately 4 msec are required to locate an address on a page of data that is already in memory, while an extra 25 msec are required if the page must be read in from the disk.

Each notify is 42 bytes long; hence a total of 6 notifies can be stored on a data base page. Searching long lists of notifies (>100 notifies, which is not uncommon for tables with 100 rows and 1 or 2 notifies per row) can be very time consuming even if all the pages are memory resident. Under conditions of heavy data base usage the search time will increase because of the high probability that the notify pages will no longer be in memory. For sufficiently long lists (>300 notifies) the list cannot be totally memory resident, and the phenomenon of page thrashing is observed. In this case searching the end of the list forces the pages with the notifies at the head of the list out of memory; the next task must then page each notify back into memory before it can proceed.

Design Improvements

We restructured the notify lists to reduce the time required to search for a notify. The major change involved grouping

[illegible] on a given row of the table together and associating that shorter list with the individual row (see Figure 1b). A long list can then be broken into many shorter lists with the following results. First each write (which can begin only at most a complete row) is followed by a search of a much shorter notify list. Also if no tasks are interested in any elements in that row, then there is no notify list to search at all! Secondly the shorter the list, the higher the probability will be that the whole list will remain memory resident, at least until the notified task can continue the search. Table I compares times for updating a parameter in a test table with notifies organized in both the old and new lists.

We also investigated methods of ordering the notifies within a list to minimize the search times. Notifies are set ultimately on a range of bytes (beg_byte_watched to end_byte_watched) within a given row. Similarly a write into the data base occurs over the range first_byte_written to last_byte_written.

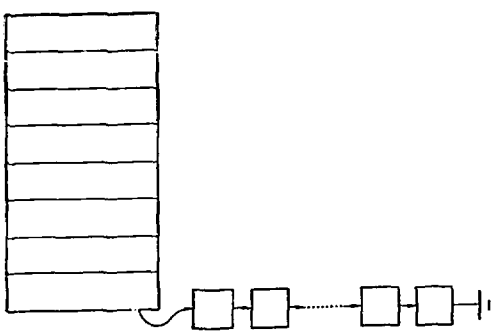

Figure 1(a). Original structure of notify list in which all notifies on a given table are arranged in a single list.

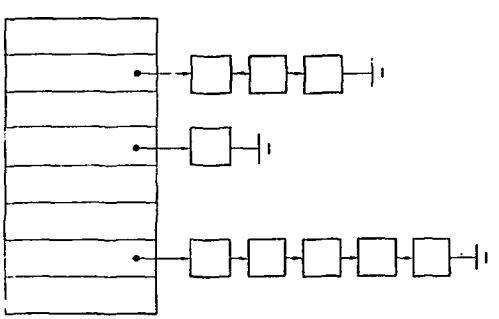

Figure 1(b). New structure in which notifies are grouped by row number and attached to individual rows of the table.

Table I. Test results for new notify lists

# of notifies on row written into	# of notifies whole table	Write and search time	
		Old lists	New lists
0	0	11 msec	13 msec
5	5	15	17
5	11	18	18
5	23	25	17
5	53	45	17
5	113	84	17
5	313	≥700	18
10	10	–	21
30	30	–	37

- Measurements made in quiescent system and do not include time required to send notification message.

We first asked whether it was possible to order the notifies so that only notifies of interest (i.e. those that must be triggered) would appear first in the list. It is possible to so order the list for a specific range of bytes written into, but there appears to be no general solution valid for all possible ranges of bytes written into. We next asked whether we can order the notifies so that all of the notifies to be triggered were located in the first part of the list, eliminating the need to search the whole list. This can in fact be accomplished by ordering the notifies in order of increasing beg_byte_watched and then in order of increasing end_byte_watched for constant beg_byte_watched. In this case we need only search the list until we find the first notify for which beg_byte_watched > last_byte_written, at which point we terminate the search (see Figure 2).The reduction of search time will of course be balanced to some extent by increases in the time required to add or delete notifies from the list. Since we expected the new list of notifies to be short, and in many cases to be completely contained on a single data base page, we decided not to implement this ordering. However such ordering ought to be valuable for much longer lists of notifies, and will be implemented if the need arises.

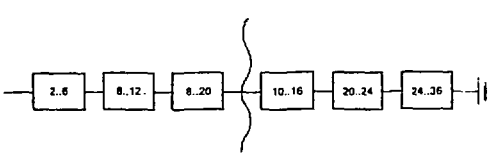

Figure 2. An ordered list of notifies in which the pair of digits indicate the range of bytes watched, and the break indicates where we can stop searching the list if we write into bytes 5 . . 8.

[illegible]

[illegible] observed a degradation in the response of the notification system which we attributed primarily to the length of the notify lists associated with a table. We investigated the notify lists by grouping tables according to the number and attaching a separate and shorter list to each row in the table. This reduced significantly the search times especially for tables with a large number of rows, and increased the probability that the list would be memory resident when the task attempted to continue the notify.

References

S. Nelson, B. G., "Notification of Change in a Database", [illegible], pp. 2049-2051, October, 1981.

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.