LA-UR -83-1370

CONF-830501--6

Los Alamos National Laboratory is operated by the University of California for the United States Department of Energy under contract W-7405-ENG-36

MASTER

LA-UR--83-1370

DE83 012674

TITLE THE Q-MEMORY RESIDENT HISTOGRAMMING SYSTEM

AUTHOR(S) M. A. Oothoudt, J. F. Amann, R. A. Floyd, J. F. Harrison, T. Kozlowski

SUBMITTED TO Real Time Computer Conf., Berkeley, CA, May 16-19, 1983

DISCLAIMER

Los Alamos Los Alamos National Laboratory
Los Alamos, New Mexico 87545

THE Q-MEMORY RESIDENT HISTOGRAMMING SYSTEM*

Michael A. Oothoudt+, James F. Amann+, Richard A. Floyd#
James F. Harrison+, and Thomas Kozlowski+

Abstract

The LAMPF Data-Acquisition System Q includes a subsystem to allow the user to create, delete, enter data in, and retrieve data from histograms stored in memory. The system is implemented on PDP-11 computers under RSX-11M V4.0 as an I/O driver with user-callable subroutines interfacing to the driver. A system with an identical user interface will soon be available on VAX computers under VMS V3.2.

Introduction

On-line storage and display of data is vital for monitoring and controlling real-time data acquisition. As a part of the Q data acquisition system (1) at the Clinton P. Anderson Meson Physics Facility (LAMPF), a histogramming system has been developed to maintain histograms in PDP-11 memory under RSX-11M V4.0. (A system with an identical user interface but different internal implementation is being developed for VAX computers under VMS V3.2.)

The histogramming subsystem is intended for use with the Q system, but is independent of it. It may be used in any application where histograms would be useful, e.g. Monte Carlo calculations.

Methodology

A software methodology developed by R. C. Tausworthe (2) was used. The methodology requires thorough documentation of the software for development and later maintenance. A requirements document is first written and reviewed by all project programmers and interested users. Then a design document and pseudo code are written and reviewed to determine the code logic necessary to satisfy the requirements. The actual code is then written, debugged, and reviewed. Finally a software specification document is written, describing the actual code for future maintenance and modification; an important section of this document gives a decision log, describing why alternative approaches were not taken.

The coding was done using structured programming preprocessors FLECS (3) and ALECS (4). The preprocessors allow writing code in a structured language containing statements similar to IF-THEN-ELSE, CASE, DOWHILE, etc. The code is then submitted to the preprocessor which converts it to a standard language (FORTRAN-II for FLECS and MACRO for ALECS). The full capabilities of the "host" language are available along with the structured constructs.

Requirements

The histogramming system was developed from a set of requirements obtained by polling the LAMPF user community. The first level of requirements maintained compatibility with the previous RSX-11D Q histogramming system:

- Ability to define at least 125 simultaneous histograms.
- Have at least 65536 bins per histogram.
- Allow both 1 and 2 parameter histograms.
- Ability to handle more than 65536 counts per bin for at least 128 bins.
- Ability to use the system outside of the Q environment.
- Subroutine access from FORTRAN to all basic histogramming functions.
- Speed of histogram entry should be comparable with the old system.
- Keep the user interface similar to the 11D system.
- Allow multiple independent users of the histogramming system.

In addition a number of new features were requested:

- Use the same or less physical memory for histogram storage and code.
- Use minimal virtual address space in the user's tasks.
- Allow histogramming any 16-bit integer on the range -32768 through +32767.
- Provide a means to control entry into histograms with a test package.
- Allow "incrementation" of histograms by integers greater than or equal to one.
- Allow entry into groups of histograms via a single subroutine call (block entry mode).
- Allow many entries to a histogram with a single subroutine call (multiple entry mode).

All requirements listed above were met. Several other requirements such as histogramming 32-bit real numbers were deferred due to lack of time and low user interest.

Functionality

The system supplies a complete set of functions for histogramming. The user's interface to the functions is through a set of subroutines that may be called from user tasks. In addition most of the functions are available through Q-supplied Histogramming Support tasks; eg. HSU allows the user to create, delete and clear histograms. These functions may be used concurrently by several tasks, so that, for example, a display program may retrieve data for plotting while an entry task is adding data to the

*Work supported by the U. S. Department of Energy.

+Los Alamos National Laboratory, Los Alamos, NM 87545.

#Presently at University of Rochester, Rochester, NY 14627.

histogram. The available functions are described below.

Startup

Task HBE initializes the system for a user. Users are distinguished by their RSX-11M User Identification Code (UIC). The current histogramming system allows up to four simultaneous independent users; a larger number of users can easily be accommodated by the code, although lack of memory for histogram storage might then be a problem.

HBE allocates storage space for histograms and makes the user known to the histogramming system. The user may choose to allocate as much space as he desires for histogram storage, subject to available memory on the computer. On PDP 11/45 computers a typical experiment uses 64 K bytes of storage.

Shutdown

Running task HOF declares that the user is no longer interested in doing histogramming and returns histogram storage back to the operating system for use for other purposes.

Peek

Task HPK allows direct inspection of the histogram storage. This is primarily a debugging function. The locations of histograms and unused storage (holes) can be listed. In addition a complete dump of the storage can be done.

Setup

Subroutine HSTSU creates histograms. The user supplies a parameter array defining the histogram limits and data to be histogrammed in histograms of one or two independent variables. Histograms may be defined at any time after program HBE has been run. The user may define as many histograms as he chooses, subject to available storage space reserved by HBE.

The total number of bins per histogram is limited to 65536. For each histogram, the user may also define up to 2048 "overflow" bins to handle cases where a histogram bin increments beyond 65536. For bins with an overflow bin associated, the maximum count is 2**32. In addition each histogram has a set of 32-bit special counters associated with it. The special counters tally entries that went out-of-range (i.e. outside the histogram limits), entries that were lost due to lack of space in the overflow table, and entries that were not made due to test failure (see below).

Delete

Subroutines HSTDEH, HSTDEB and HSTDEA "undefine" or delete histograms. After being deleted a histogram can no longer be accessed and its storage space is returned to the histogramming system for use in setup of new histograms.

Clear

Subroutines HSTCLH, HSTCLB and HSTCLA clear a histogram. After clearing, a histogram still exists but all bins, overflow bins, and special counters are set to zero.

Entry

Subroutine HSTBLK enters data into histograms via a "block entry" mode. Blocks of histograms usually correspond to logically related data. For example, in a two arm coincidence experiment three separate blocks might be used: one for each arm's singles data and one for the coincidence spectra. The user's code computes a number of variables to be histogrammed and stores them in an array. When HSTBLK is called, the user specifies which group (block) of histograms the array corresponds to. For each histogram in the block, the histogramming system extracts the data from the array and calculates the corresponding bin in the histogram. It then adds a user-supplied integer (usually one, but any integer 1 through 32767 is allowed) to the bin. Note that with this block entry mode, the user can define new histograms for a block at any time and, if the data to histogram is already in the array, the user need not modify his code.

The entry function is the only histogramming function that need be done on an event-by-event basis, and thus it is the only time critical function in the histogramming system. Timing tests on a PDP-11/45 show that the entry speed (in milliseconds) is approximately

$$T = 1.64 + 0.125*N$$

where N is the number of histograms in the block. The constant term is primarily due to the operating system QIO overhead while the linear term is primarily due to the time needed to find the histogram parameters, convert the user's data into a histogram bin number, and make an entry in the bin.

HSTBLK also allows a "multiple entry" mode in which many entries to a single histogram may be made with a single subroutine call. This is useful, for instance, in monitoring wire chambers where a single event may cause many wires to be hit. Multiple entry may also be used for high speed histogramming: Data may be accumulated in local task memory until the buffer fills or there is a beam-off period; the data may then be passed to the histogramming system. The speed for this mode is somewhat faster since the histogram parameters need be found only once for several entries:

$$T = 1.69 + 0.075*M$$

where M is the number of entries into the histogram. With this technique histogramming entries may be made in excess of 10000 per second on a PDP-11/45.

Entry Initialization

Subroutine HSTINI must be called by any task which will call subroutine HSTBLK. HSTINI initializes a local data base used by HSTBLK to improve entry speed.

Entry Testing

The histogramming system allows the user to dynamically determine whether data should be entered into the histogram. An array of logical variables is used to control entry. At histogram setup time, one of the histogram definition parameters is the number of the array element to be used to control the histogram. At entry time, that array element is checked, and if its value is logical true, the histogram entry is carried out. The Q test package (1) may be used or any user testing system that generates a logical test result array may be used.

Retrieval

Subroutine HSTOWN allows the user to retrieve a list of users of the histogramming system.

Subroutine HSTHPB allows the user to retrieve the definition parameters for any currently defined

histogram.

Subroutine HSTRET allows the user to retrieve the data and special counters for any histogram.

Set, Update

Subroutine HSTUPD allows the user to add an array of data to a histogram, with each array element being added to a single bin of the histogram. The subroutine allows the user to either Set the bin to the array value (by zeroing the bin before adding) or Update the bin by adding the array value to the current contents.

This option is useful for restoring archival data to memory. It may also be used for high speed histogramming if the user accumulates a "histogram" array in his task and then passes it on to the histogramming system in beam-off periods.

System Implementation

A diagram of the functional structure and data flows of the histogramming system is shown in Figure 1. Functions inside dashed borders are separate tasks; functions inside dotted borders are subroutines that may be called by Q tasks or user tasks. Data flows are shown by dashed lines (low speed) or double dashed lines (high speed required). Direction of data flow is shown by arrows. Error return and parameter passing paths are not shown. Communications with the IO driver HM: is via QIOs except for HM:'s direct connection to the memory management region.

The three tasks and 13 subroutines discussed above are the user's basic interface to the internals of the system. (Another set of optional Histogramming Support tasks are described below). For the vast majority of applications, the user need know nothing about the internal structure of the system.

The core of the histogramming system is the IO driver HM: and the memory management region which contains the histograms. HM: controls and synchronizes all access to histogram data. An I/O driver was chosen for the PDP-11 histogram system to allow tasks easy access to more histogram storage than is available to a single task. Other implementions could have been chosen, but an I/O driver takes advantage of a standard operating system interface to privileged code. Furthermore a driver by its nature provides synchronization between histogramming operations in different tasks and is reasonably fast. HM: receives all histogramming requests, processes them, modifies the MM region as necessary and returns requested information and error status. The I/O driver is approximately 4 K bytes long and approximately 250 bytes of code are required in the user task for histogram entry. Further space required for histogram storage is allocated by the user's running HBE.

Histograms are stored in an RSX-11M Memory Management region. Each separate user (UIC) has a separate region. User tasks access the region through the driver only and thus need know nothing of the details of memory management or data storage architecture. A standard memory management region was used in spite of its awkward interface to prevent conflicts with the operating system's memory allocation. The region consists of a linked list of holes for unused memory and a list of histograms. For each histogram a set of histogram definition parameters, an overflow table, and the histogram bin data are kept.

The system accesses a histogram through a series of tables. The Owner Table is a list of users (UICs) who have run HBE. For each owner in the table, a

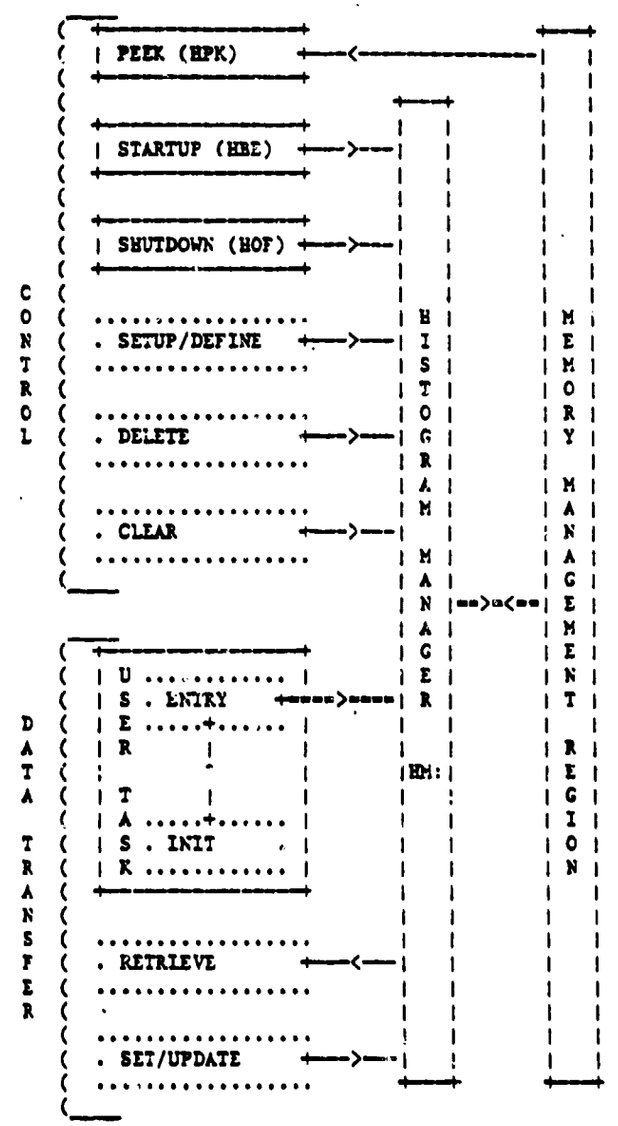

Figure 1 - Overview of Histogramming Data Flows

pointer is kept to a Block Table. The Block Table contains an entry for each histogram block the user has defined; the entry is the address in memory of the first defined histogram in the block. Each histogram in turn contains a pointer to the next histogram in the block.

Thus to access a histogram, HM: first determines which owner (UIC) has made a request. From the Owner Table, it can then go to the Block Table. If a block number was given in the request, HM: checks each histogram in the block for the requested one. If no block number is given, every histogram in every block must be checked to find the desired one. This algorithm is somewhat inefficient, but most histogramming functions are not speed critical enough or used frequently enough for this to be a problem. For the block entry function, the algorithm is very good since the user must specify the block number in the call to HSTBLK and every histogram in the block will be accessed in any case.

Support Systems

In principle the system above is adequate to provide all the user's histogramming needs. In

practice a support system is needed to take care of common experimental needs such as archival storage of memory-resident histograms. The Q Histogram Support Subsystem provides these services through tasks which call the histogramming subroutines. The available tasks are

- HLI—Lists definitions for histograms in memory or archival storage.
- HPL—Plots one- or two-parameter histograms on Tektronix 4010 compatible display terminals.
- HPR—Prints histogram contents.
- HSU—Sets up histograms from input given by the user at the terminal keyboard or from a file. This allows the user to dynamically change his histogram set. The task also allows deleting or clearing histograms from the keyboard.
- HSV--Saves memory-resident histogram data on disk for archival storage.

Conclusions

The Q histogramming system is projected to satisfy the needs of the user community for the expected lifetime of our PDP-11 computers. Its modular construction should make adding new features relatively easy.

References

1. J. F. Harrison, T. Kozlowski, R. A. Floyd, J. F. Amann, G. T. Anderson, M. A. Oothoudt, and D. G. Perry, "Design of the RSX11M Q System," IEEE Transactions on Nuclear Science, Vol. NS-28, No. 5 p. 3724-3730, Oct. 1981

2. R. C. Tausworthe, "Standardized Development of Computer Software," Englewood Cliffs, NJ, Prentice-Hall Inc., 1977, Vol. 1 and 2

3. T. Beyer, "FLECS: User's Manual," Department of Computer Science, University of Oregon, 1975 unpublished

4. M. A. Oothoudt and T. Kozlowski, "ALECS: Assembly Language Extensions and Control Structures," Proceeding of the Digital Equipment Computer Users Society p. 537-540, Dec. 1982