Published: 9/28/66

Identification

Library Procedures Used by the Interactive Debugging Aids
D. B. Wagner

Purpose

Section BY.6 describes a collection of procedures used in the implementation of the interactive debugging aids described in BX.10.00 - .04, and as such constitutes the bulk of a design specification for the debugging aids.

The diagram of figure 1 shows the interaction among the procedures described and their intended use in the debugging aids. It is to be hoped that these procedures will be useful in other areas as well, particularly in the construction of an interactive "desk calculator" command.

Expression Evaluation

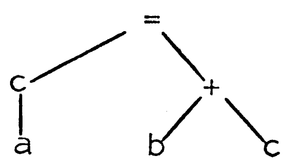
The user will see the debugging aids primarily in terms of the expression language, so that expression-evaluation is the most sensitive area of the implementation. Such issues as the order in which symbol tables are searched (e.g., which _a_ will be found when there exist more than one _a_) and the points where extra blanks may occur seem minor, but have a great effect upon the user - they determine the "personality" of the debugging aids.

The work of expression-evaluation is done by three procedures - _parse_, _evaluate_, and _setvalue_ (described in BY.6.03, BY.6.04 and BY.6.05 respectively). _parse_ performs the syntactic analysis, yielding a tree-representation of an expression. _Evaluate_ and _setvalue_ perform, respectively, the two kinds of semantic expression-analysis, the so-called "right-hand-side" and "left-hand-side" evaluations, named for the two sides of an assignment statement. _Evaluate_ finds the value of an expression, and _setvalue_ sets the value of an expression.

For example if the request

    set c(a) = b+c

(meaning set the contents of location _a_ to the result of adding _b_ and _c_ together) is typed to _probe_, the following action takes place. _Parse_ is called by _probe_. It produces a tree-representation of the given expression as follows:



Then probe gives the "right-hand" subtree

$$\begin{tikzcd} & + \arrow[dl, dash] \arrow[dr, dash] & \\ b & & c \end{tikzcd}$$

to evaluate, which looks up b and c in the symbol table, adds their current values together, and returns the result. This result and the "left-hand" subtree

$$\begin{tikzcd} c \arrow[d, dash] \\ a \end{tikzcd}$$

are given to setvalue, which actually changes the contents of location a.

Handling of Symbol Tables

The standard format for the "segment symbol table" is specified in Section BD.1. What is important about this standard format is that it allows considerable latitude to the designer of a translator in the area of descriptions of symbols, but rigidly legislates enough about the symbol table to make it possible for a standard subroutine to search a table for a given symbol-name without reference to what translator produced the table.

Since many translators (including PL/I and the standard assembler) use block-structuring to determine the "scope" of symbols, the symbol-table format allows for tree-structuring. The scheme used allows for tree-structuring at a higher level as well: so that all the individual symbol tables known to the debugger are treated as subtrees of a single tree called the Combined Symbol Table. Two procedures are provided for maintenance of the Combined Symbol Table: find_table and lose_table (described in BY.6.02). Find_table locates the symbol table for a specified segment and makes it a part of the combined Symbol Table (makes the symbol table "known"). Lose_table eliminates the symbol table (makes the symbol table "unknown").

The procedure Search_tables (described in BY.6.02) is used by evaluate and setvalue to search the Combined Symbol Table. It starts at a specified point and searches up and down in the tree according to certain rules which are discussed in BY.6.01.

The Request Dispatcher

The Request Dispatcher described in BY.6.01 does most of the work of reading and recognizing requests for the debugging aids, and in this respect bears some resemblance to the Shell. It also handles the _if_, _else_, _do_, and _end_ requests, common to all of the debugging programs, which provide conditional and replication facilities.

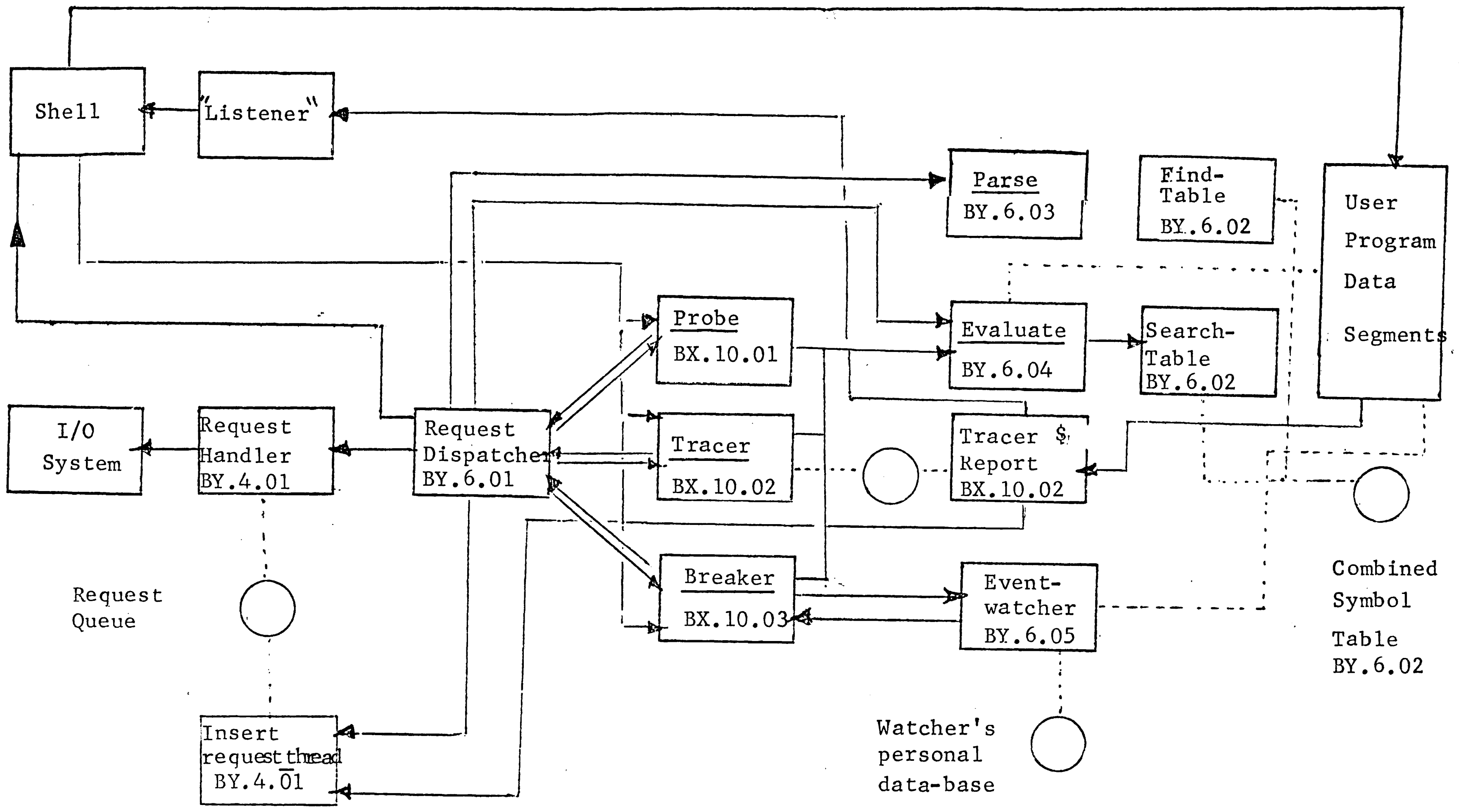
To use the Request Dispatcher, an interactive program calls the entry dispatch_request with a list of expected request names and a corresponding list of procedure entry points telling what procedure is to perform each type of request. The Dispatcher calls the Request Handler (see BY.4.01) to obtain one request line, extracts the request name (the first identifier on the line) and looks it up among the request names given in the call. If the request name is found there, dispatch_request calls the routine specified to perform the request. When this routine returns, dispatch_request returns to its caller.

If the request name is not found in the given list, it may be one of the special control requests _if_, _else_, _do_, and _end_. If it is, the appropriate action is taken (this involves fiddling with the Request Queue, see BY.4.01). If not, the request is either a command or an error. The line is given to the Shell, which attempts to interpret it as a command.

The Watchers

The "watchers" are the routines used by the _breaker_ command to arrange for notification upon the occurence of events of interest to the user. Their use bears a family resemblance to the use of the _on_ and _signal_ statements in PL/I: a call to a watcher specifies an event to be watched for and gives an action to be performed when the event occurs, the "action" in this case being simply a call to a procedure with a single "identification" argument plus any arguments necessary to describe the event completely.

Not very much can be said about the watchers right now since not very much is known about the System fault and interrupt handling machinery.


Shell
"Listener"
Parse BY.6.03
Find-Table BY.6.02
User Program Data Segments
Probe BX.10.01
Evaluate BY.6.04
Search-Table BY.6.02
I/O System
Request Handler BY.4.01
Request Dispatcher BY.6.01
Tracer BX.10.02
Tracer $ Report BX.10.02
Breaker BX.10.03
Event-watcher BY.6.05
Request Queue
Insert request thread BY.4.01
Watcher's personal data-base
Combined Symbol Table BY.6.02