Published: 6/7/66

Identification

Program tracing under interactive control
tracer
D. B. Wagner

Purpose

Use of tracer allows the execution of a program to be monitored on-line in as fine or coarse a manner as desired.

Usage

To use tracer the user inserts at strategic points in his program calls (using the standard call sequence) to the entry tracer$report in the tracer command. Some ways of causing these calls to occur automatically on occurrence of certain events will be provided, such as the breaker and monitor commands described in BX.10.03 and BX.10.04. There may also be a debug mode in the PL/I compiler which causes such calls to be inserted between statements, giving full information concerning variables changed, previous and new values, etc. The tracer command is then used to store up actions to be performed whenever tracer$report is called with certain arguments. These actions may include both commands and requests to commands.

The format of the calls to tracer$report is essentially up to the user, except that the first argument must be a character-string name for the call which will be used to identify the actions to be performed.

The command

tracer

causes tracer to begin reading requests from the console. The user may type any of the requests listed below or any of the "control" requests (if, else, do, end) described in BX.10.00. He may also type macro invocations (in the same form as in the command language: see BX.1.01) which expand to sequences of these requests. If a line received by tracer (after macro expansion) is not recognizable as a request, it is treated as a command. The line is given to the Shell, which gives an appropriate diagnostic if it is not a command either. *

Implementation *

The following digression is necessary to explain the action of the tracer command. See the diagram of Fig. 1. The command listener, the debugging programs, and most other

Interactive programs read their requests through a "request handler" which acts as an interface to the I/O system. It expands macros, handles the semicolon convention, etc. The request handler keeps a special data base called the request queue, and before the request handler reads a line from the console it checks to see if there are any lines waiting in the request queue. If there are it uses the first line in the queue instead. (The macro processor will be one program which places lines into the request queue: the entire first-level expansion of a macro invocation is simply put at the head of the queue.)

When the setaction request described below specifies a name and a list of commands and requests, these are stored in the tracer data base. When eventually the user's program is started and a call to tracer$report occurs, the first argument of the call is matched against all the stored-up names. If a match is found the corresponding list of command and request lines is placed at the head of the request queue and the command listener is called. This scheme provides a very general tracing facility.

Requests to Tracer

Setaction and endaction are the basic requests: the sequence

```
setaction name
   .
   .  action
   .
endaction
```

causes the action specified to be stored away in a data base used by the trace entry. The name is a character-string identifier, to be matched against the first argument of each trace call. Action is a sequence of commands and requests. It is stored up to be performed whenever a call to the trace entry tracer$report has the first argument equal to name. If more than one action has been specified by setaction requests for a given name, they will be performed in the order given. Name may be "*", in which case action is to be performed on every call to tracer$report.

The action specification may of course include conditional (if ... then ...) requests which narrow down the selection of action still further than the naming convention does. Expressions may include the special function

```
tracearg(n)
```

which gets the n'th argument of the last trace call.

The request

resetaction name

deletes all actions stored up for the name given.

The requests

listaction
listaction name

cause either all currently specified actions, or all currently specified actions for the given call name, to be listed in a convenient format.

The request

exit

causes tracer to return to its caller, usually the Shell.

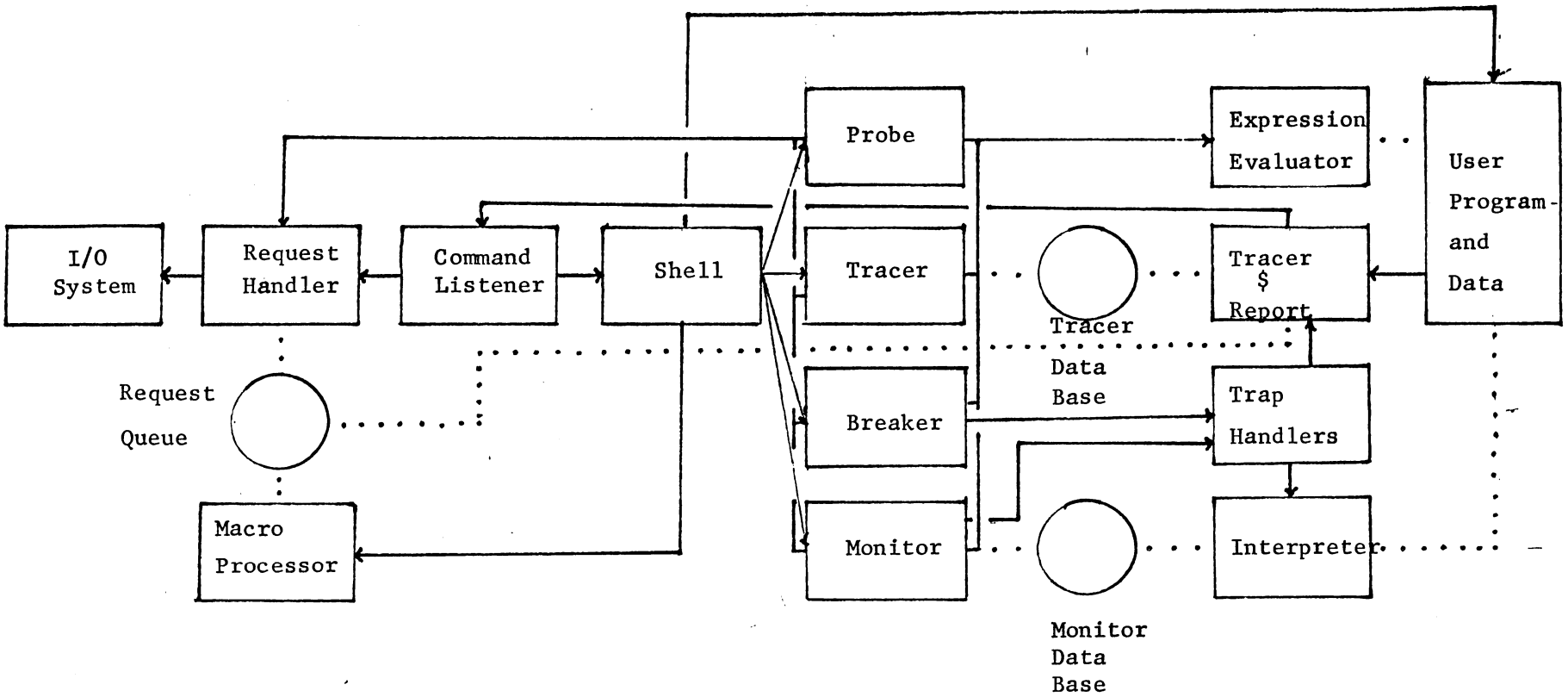

Figure 1: Probe, tracer, breaker, and monitor and their relation to selected parts of Multics. Arrows represent calls and dotted lines represent data paths.