Draft for approval
Published: 4/8/66

Identification

Trace and Dump for EPL procedure debugging in the 645 simulator environment.
J. Ridgeway

Purpose

To provide some early tools to assist in process debugging in the simulator environment. The initial version provides facilities for:

1. Tracing procedure calls
2. Tracing inter-segment references
3. Dumping segments in various formats
4. Recording user supplied comments on process activity

Description

The user debugging a process will typically write a process driving procedure that simulates the process environment by generating a data base and issuing procedure calls. The user includes the debugging aid in his process to monitor the progress of procedures and preserve machine conditions at junctions in the process.

The debugging aid consists of essentially three components which are dumper, trace setup, and a working data segment (generated by the trace setup) called trace data base (fig. 1). The dumper may be called at anytime from any procedure in the process, e.g., after a procedure return to dump the data base affected by the procedure. The trace setup must be called before any segment to be included in the Trace is linked to segments in the process, these calls will probably reside in the process driving procedure.

When any procedure of a process is in execution and it references a segment external to itself a linkage fault occurs. These faults invoke the tracer. The segment that is external to any procedure in the process is considered to be 'unlinked' to that procedure until it is referenced a first time by that procedure. A segment can be 'unlinked' to procedure A of a process even though it is linked to procedure B of that same process. Procedure references to internal locations can not be traced because the tracer is dependent on linkage faults. When the user issues a request to trace, a call is made to the trace setup with the arguments of the call defining the reference or call to be traced. The trace setup copies these definitions into the trace data base and sets a flag to indicate to the LINKER that tracing is in effect. Thereafter when a linkage fault occurs because an unlinked segment is referenced or called, the LINKER passes control to the tracer before completing the linkage. The tracer extracts the names of the procedure or data segment being referenced from the linkage and definitions of the referencing procedure. A comparison is made

between these names and names in the tracer data base supplied by calls to the tracer. If the event is to be recorded the tracer writes the necessary information on a trace output file, modifies the callers linkage so subsequent references will be intercepted and the process is resumed.

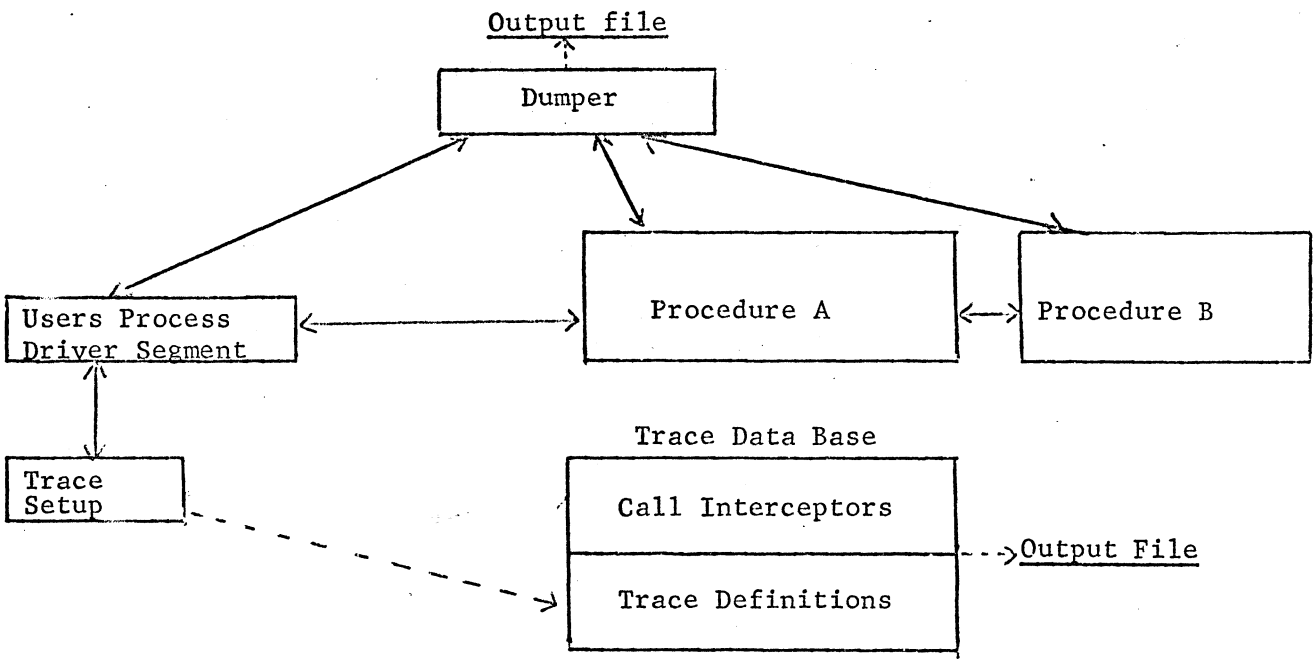

Figure 1

The Trace and dump output are recorded on the Trace output EPL file which is returned to the CTSS files or listed on the 635 printer by the 64.5 dumper. This file is written in a format compatible with EDA or PRINTA.

The debugging aid is incorporated into the simulator environment by extending the LINKER procedure. This reduces the implementation effort and simplifies usage. The debugging aid is always resident with a process being simulated because it is part of the LINKER (which is a special inclusion segment).

Trace and Dump Calls

The first two calls are for defining the segments to be traced and must be executed before the procedures or data segments of interest are linked to any procedures in the process. The remaining calls do not depend on linkage faults and can be interspersed throughout the users process.

A. Trace procedure calls

```
call linker$tracecall ('procname$entry', number of arguments to be
dumped, argument number, format, dimension,...)
```

When the procedure specified by the character string argument is called from some other unlinked procedure the tracer intercepts and records the occurrence of the call and dumps specified input arguments. All calls to this procedure will be intercepted for the duration of the process.

Argument 1 - Name of procedure to be traced. Argument passed as character string expression. If no entry point is specified the entry procname$procname is assumed. If the procedure name is 'all' all unlinked calls will be recorded with no arguments being dumped.

Argument 2 - number of arguments to be dumped. Specifies the number of input arguments to the called procedure that are to be dumped on the output file on each call.

Argument 3i - Argument number of input argument to be dumped.

Argument 4i - Format of input argument. Specifies the Trace output format and directs the tracer in its interpretation of EPL data specifier and dope vectors.

Format Specifier Value	Output format
1	signed decimal integer
2	octal number
3	character string
4	bit string
5	signed decimal integer-72bit
6	octal number - 72bit

Argument 5i - dimension value. Any non-zero value signifies to the tracer that the argument is an array. If the array is one dimensional the value will be used as the number of elements to be dumped. (If the tracer finds that the value of this argument is greater than the value computed from the upper bound and lower bound found in the dope vector for the array the dope vector values establish the dump limit). If the input argument to the called procedure is not an array the dimension must be zero.

B. Trace inter-segment references

```
call linker$traceref ('segment$symbol', n,...)
```

When the unlinked segment and symbol is referenced the tracer intercepts the reference, records the occurrence of the reference and dumps the octal value of the instruction in execution and the AQ registers.

Argument 1 - name of segment reference to be traced. Argument passed as character string expression. If the user has access to an assembly listing of his procedures he is given the facility to be more explicit in his trace definitions. The user may specify all of the items found in the assembly language address field.

Example 'sega$symbx ± exp,m'

Where exp must be a decimal integer and modifier is any legitimate assembly language modifier.

If the name is 'all', then all inter-segment references are recorded.

Argument 2 - the first "n" times the reference is to be traced during the process.

C. Terminate Trace

```
call linker$endtrace
```

Causes the tracer to remove current trace definition from its data base and repair any linkage segments it has modified. All inter-segment references and procedure calls that would have been linked if the tracer had not interfered become linked. Subsequent traces of unlinked segments can be established by the issuance of calls described in A. and B.

D. Set file size limit

```
call linker$setfile (n)
```

This call establishes a limit on the size of the tracer output file. Trace and dump output are terminated after the file length exceeds "n" CTSS Tracks (432 words per Track). The default value is 50 CTSS Tracks.

E. Dump

```
call linker$dump ('segname$symbol', format, length, dimension, ...)
```

The specified segment(s) is (are) dumped in the requested format on the trace output file.

Argument 1. Name of segment - passed as a character string expression. If only one name appears it is assumed to be the name of the segment (e.g. 'segname')

Argument 2. Output format specifier. Available formats in the initial version are as follows:

Format Specifier Value	Output Format
1	signed decimal integer
2	octal number
3	character string*
4	bit string*
5	signed decimal integer-72bit
6	octal number-72 bit

Argument 3. Length of data element - no. of items per element (i.e. bits/string)

Argument 4. Dimension - number of data elements to be dumped beginning with 'segname$symbol'.

*dumper assumes packed non-varying strings whose length is defined in argument 3.

F. Remark

```
call linker$remark ('any character string')
```

The specified character string is written on the tracer output file. This call allows the user to record the progress of his process by writing comments as he reaches junctions in the process. It further provides a method of identifying an area of the output file that can be searched for selective listing by EDA.

Output Format

The output from the debugger is written on an EPL file and returned to the users CTSS files or listed on the 635 printer by the 64.5 dumper. The format of the file is compatible with EDA or PRINTA. Each type of line in the file has a unique character string at the beginning of the line so that selective listing

can be done with EDA.

The file format is as follows:

A. Dump format

```
dumpident   sega$symx
dump   000002    xxxxxxxxxxx...xxxxxxxxxxx
 .
 .
 .
dump   000100    xxxxxxxxxxx...xxxxxxxxxxx
```

Where the first line identifies the dump block and is a copy of the call dump argument. The first 6 characters after the subsequent line headings indicate the element count. This count has an octal radix for octal format dumps and decimal for all others.

B. Remark format

```
remark   aaa...a
```

Where 'aaa...a' represents the character string supplied by the user in the call

C. Call trace format

```
call proca$entry1  from procb$loc

arg   arg#, value ..., arg#, value
```

D. Inter-segment reference trace format

```
extref    sega$symx    from    segb$loc

mach    instr = xxxx xx     aq = xxxxxxxxxxxx,xxxxxxxxxxxx
```

Where the xxxx xx after instr represent the octal value of the instruction and tag fields of the instruction in execution when the linkage fault occurred. The X's after the aq represent the octal value of the aq registers.

E. When tracing is terminated because (a) "endtrace" is called or (b) the output file has more words than specified in a call to "setfile", the output message is as follows:

```
        end trace called  -  tracing terminated
```

If tracing is terminated because of output file overflow subsequent calls to Trace or dump routines will be ignored and control returned to the calling procedure.

F. Error messages

error trace call error 'call arguments'

If an error in the arguments to a trace or dump routines occurs, the error is flagged and identified on the output file. The call in error is ignored and control is returned to the calling procedure.

Possible Future Modifications

1. Tracing procedure returns and dumping specified output arguments and timing information.

2. Providing return arguments when calls are trapped so missing procedures do not have to be coded and debugged as dummy procedures.

3. Providing procedure calls when specific inter-segment references or procedure calls are trapped.

4. Integrating the Trace output file into GEBUG activities.