Published: 11/10/67

Identification

Strace: a subroutine tracing procedure for 6.36
D. B. Wagner

Purpose

A procedure has been written as a by-product of debugging the debugging aids which greatly aids in debugging 6.36 programs. It provides a very simple-minded version of the service provided by itsda (BE.12.01) and works.

Usage

Strace is named after the CTSS command STRACE, to which it bears a certain resemblance. Strace catches calls to specified entries, reports certain details of such calls, and lets them go ahead. To use strace the user should write an EPL program which includes the following statements for each entry x he wishes watched. Execution of this program then causes watching to begin.

```
     dcl x external entry;

     call strace (x, message, limit);
```

Here message is an arbitrary character-string to be included in each report of a call to x and limit is the maximum number of reports desired for this entry.

The report in the error file that an entry has been called will have the following format:

```
     ... user's message

     ... n'th call to <segno>|loc.  From <segno>|loc

     ...
```

where n is a decimal number and the seg's and loc's are octal numbers.

Segments Needed

The following segments are needed in using strace. They exist on the Multics Segment Library.

Segment Name	Descriptor
derail_catch	(MASPRC,SLVACC,WPERMT)
make_impure	(MASPRC,SLVACC)
strace	(SLVPRC,SLVACC)
watch_callto	(SLVPRC,SLVACC)
watch_instruction	(SLVPRC,SLVACC)
watch	(SLVPRC,SLVACC)

Implementation

Strace is a very simple program which uses some modules of the interactive debugging aids, principally the "event_ watchers" watch_callto and watch_instruction. Since users may wish to make up special debugging aids of their own, these procedures are described below. An overview will be found in BY.6.06, but these particular watchers are not yet described there.

To watch for control passing to a given location in a procedure, the call is:

```
     call watch_instruction (callback,id,cp);
```

where the arguments expected are declared,

```
     dcl callback entry returns (bit(1)),

          (id,ip) ptr;
```

Here callback is the caller's agent, to be called by watch_ instruction when the event occurs. Id is a pointer used by the caller to keep his watching straight. Ip points to the instruction to be watched.

Watch_instruction saves the instruction and plants a drl (derail) instruction in its place. It stores away all the information it needs in a personal data base and returns.

When control reaches an instruction being watched, a derail fault occurs. Control passes to an entry in watch_instruction which does

```
     b = callback (id,mp);
```

with the declarations,

```
     dcl b bit 1,

       (id,mp)ptr;
```

Here callback and id are as in the call to watch_instruction. Mp points to machine conditions in standard form.

If callback returns "1"b, watching of the instruction is to be continued. If it returns "0"b then all watching associated with this id is to be discontinued.

To watch for a call to a certain entry, the call is:

```
     call watch_callto (callback,id,ep);
```

where callback and id are as above and ep is a pointer to the entry.

Watch_callto uses watch_instruction to watch for control passing to the given entry. When the call occurs it does a

```
     b = callback (id,mp,cp,ap);
```

with declarations,

```
     dcl b bit (1),

          (id,mp,cp,ap)ptr;
```

B, callback, id, and mp were discussed above. Cp points to the caller of the watched entry and ap points to the argument list of the call.