SLIP traps present a relatively non-code modifying method to gather additional diagnostic data to aid in MVS/XA/ESA problem solving.

This series has examined using IPCS for MVS/ESA Version 5 (Part I, Technical Support, October 1996); Part II (November 1996) presented a technique called one-page MVS/XA/ESA abend analysis; and Part III (December 1996) delved into advanced SVCDUMP analysis. This month’s article will examine, in detail, SLIP traps and GTF tracing. The examples and discussion of keywords will help you effectively develop SLIP traps. I will also show you how to debug SLIP traps when they don’t work as expected.

SLIP (Serviceability Level Indication Processing) traps allow us a relatively non-code modifying method to gather additional diagnostic data. Many MVS/XA/ESA problems either do not create the diagnostic data we need (dumps and GTF traces) or create this information after we need it. For example, a SLIP trap can be set to take a diagnostic SVCDUMP before any MVS recovery routines have executed. We get the information before any recovery may have altered anything critical to our analysis.

There are two general categories of SLIP traps: non-PER traps, which are used for system errors, and PER (program event recording) traps that use microcode (control registers) to allow SLIP traps to monitor instruction fetch (IF), successful branch (SB1 or SBT) or storage alteration (SA or SAS).

There are two types of program event recordings (PER) for ESA. PER 1 is for ESA/370 and PER 2 is for ESA/390. PER 2 will allow the option of an event to occur within designated address spaces for storage alteration or successful branch. Figure 1 shows the layout of control registers 9, 10 and 11 used in a SLIP PER trap.

SLIP TRAPS AND SYS1.PARMLIBS

You can either enter SLIP traps through the SLIP subcommand of the TSO OPERATOR command or use the SYS1.PARMLIB members IEACMD00, COMMNDxx or IEASLPxx member (where xx is the member suffix). I prefer to use the IEASLPxx member since you can use multiple lines for one SLIP command. See Figure 2. For non-PER SLIP traps, you can:

- create a tailored SVCDUMP to your specifications;
- suppress unnecessary dumps (such as any...
I caused repeated SVCDUMPS on my trap which was highly embar -

...ing them. If you do not spec -

...t trap is the ID field of your SLIP trap), you can see the default MATCH -

...y, which is common with DB2

Son=04. The proper use of the DATA trap keyword is for a register 15

Son=04. You could also specify DATA=(15R,EQ,04) instead of REA -

...tr ap will limit this SOC4 condition to job BATCHJB1 only. The com-

...trace every time the trap is sprung. ACTION=RECORD can only be

...specified for non-PER SLIP traps. Note that your EREP entry will not

...any symptom data (from the SYMREC macro) to assist in the debugg-

...ncluding the address range of X’007E’ to X’008E’ for SLIP trap consid-

...load module MYPGM3. All PER SLIP traps include instruction fetch (IF),

...for example, PVTMOD=(MYPGM3,007E,008E), which would include the

...traps are commonly used to see if code is being executed.

...the following restrictions to ACTION=SYNC-SVCD are that the PER ad-

...locked; and

...and in taskmode.
In the following SLIP PER trap example if we are executing LPALIB- resident load module IGC0008A, then the trap will spring. The use of the LPAMOD keyword will tell MVS we want IGC0008A to be a LPALIB- resident load module. If we wanted a nucleus module we would specify NUCMOD. We can specify valid storage address ranges for PVTMOD, LPAMOD and NUCMOD keywords as PVTEP, LPAEP and NUCEP respectively. For example:

```
SLIP SET,ID=TJB9,
  JOBNAME=PDCELA,
  LPAMOD=(PTLDRVRM,BB28),
  PVTMOD=(PTLDRIVM,BB28),
  ACTION=SVCD,
  MATCHLIM=1,
END
```

While I have not yet needed to work up a SLIP PER trap for a successful branch. I have worked up a trap for a storage alteration event. Storage alteration problems are highly irritating to system users and can cause MVS to go down immediately if the storage alteration is deep within MVS. See Figure 4.

The SA keyword after SLIP SET indicates that this is a storage alteration PER trap. The PVTMOD specification in the storage alteration trap is where the storage alteration is coming from (what program) and the RANGE keyword indicates what address range that PER will monitor. The RANGE keyword is only for SLIP PER traps. The PRCNTLIM keyword parameter indicates that a software limit (about 34 seconds between PER interruptions) must occur before a trap will be disabled because of this limit. When you set your SLIP PER trap, you may see some of these messages. After you use the SET SLIP=XX (where XX is the beginning messages. After you use the SET SLIP=XX (where XX is the beginning

```
SLIP SET,ID=TJB8,
  JOBNAME=TB,
  IEES72I SLIP TRAP ID=TJB8 SET BUT GTF IS NOT ACTIVE
  IEES36I SLIP VALUE TB NOW IN EFFECT
```

The special keyword DEBUG will instruct GTF to create a standard trace entry (we need to specify ACTION=TRACE and TRDATA=(STD,REGS) to get this information. See Figure 7.

```
DEBUG,
```

The modifier of .BRYANT will allow me to later shut down my GTF procedure by issuing the MVS command P BRYANT instead of a tape and/or DASD device used as the stepname when no modifier is present. We would respond U to the AHL125A trace options command shown in Figure 6. As stated before, the P BRYANT command in this case would shut down GTF.

```
S GTF.BRYANT, MEMBER=GTFSLIP
```

```
SLIP SET,ID=TJB9,
  JOBNAME=MYLOOPJB,
  TRDATA=(STD,REGS),
MATCHLIM=100,
END
```

In the above SLIP PER IF trap example, the keywords of note are ACTION=TRACE and TRDATA=(STD,REGS). For our ACTION of TRACE (GTF tracing) we want the standard GTF SLIP trace record (STD) and the general registers (REGS) for each time the trap springs. I feel requesting the general registers is a must, since you will probably be working with the GTF SLIP trace records, general registers and a (later) storage dump of the given problem. ACTION of TRDUMP will instruct the system to perform GTF tracing and schedule a SVCDUMP when the trap completes or is disabled for any reason.

When you set the SLIP PER trap with GTF tracing (in our previous example, ACTION=TRDUMP), MVS will set the trap and issue the messages shown in Figure 5. Our GTFSLIP trace PARMLIB member has specified TRACE=SLIP. I use a procedure called GTF as follows:

```
S GTF, MEMBER=GTFSLIP
```

```
SLIP SET,ID=TJB8,
  JOBNAME=TSOID,
  ACTION=TRACE,
  TRDATA=(STD),
  COMP=04,
  REASON=08,
  MATCHLIM=1,
END
```

The modifier of .BRYANT will allow me to later shut down my GTF procedure by issuing the MVS command P BRYANT instead of a tape and/or DASD device used as the stepname when no modifier is present. We would respond U to the AHL125A trace options command shown in Figure 6. As stated before, the P BRYANT command in this case would shut down GTF.

```
S GTF.BRYANT, MEMBER=GTFSLIP
```

The special keyword DEBUG will instruct GTF to create a standard trace entry (we need to specify ACTION=TRACE and TRDATA=(STD) to get this information. See Figure 7.

```
Our PARMLIB member contains the TRACE value of SLIP. In our example, I execute a program to get an SOC4 interrupt code 4. From the SLIP TRAP DEBUG output, look at the first two digits
It’s a good idea to research the abend you will trap because many MVS abends really started out as S0C4 abends in system code.

immediately below the PKM value (in our case this is a hex 16) The hex 16 is decimal 22 which we look up in Figure 8 (from the manual MVS/ESA Diagnosis: Tools and Service Aids and find out what the decimal 22 indicates about our trap situation. In our situation, we have a mismatch on the reason code of 08 we specified on the trap and the 04 reason code from the actual program. This problem was easy to fix, although if you have multiple problems with a trap you will need to do this multiple times. SLIP DEBUG catches only one error at a time. As always, it is best to create your toolkit ahead of time to avoid incurring additional problems.

**SLIP TRAP RECOMMENDATIONS**

Always make a trap as specific as possible and specify ID, JOBNAME, ACTION and MATCHLIM wherever possible. It’s a good idea to research the abend you will trap because many MVS abends really started out as S0C4 abends in system code. MVS recovery routines change the abend code to something more meaningful. The examples in this article will be a good starting point for creating your diagnostic toolkit.

Part V, the concluding article in this series, will cover the MVS standalone dump which most systems programmers don’t like to think about. Hints and tips will be provided to help you analyze the problem quickly.