Session 17093
SLIP Trap Bootcamp
Part 2

z/OS Systems Programming & zNextGen – March 3rd, 2015

Patty Little plittle@us.ibm.com
IBM Poughkeepsie

SHARE
Technology · Connections · Results

Trademarks

The following are trademarks of the International Business Machines Corporation in the United States and/or other countries.

- **MVS**
- **OS/390®**
- **z/Architecture®**
- **z/OS®**

* Registered trademarks of IBM Corporation

SLIP
Examples

Case 1

Key points:

Using COMP/REASON to qualify error

Using JOBNAME to qualify address space

Using ADDRESS to identify where error must occur

Case 1

- Some unidentified code is **bad-branching to address 5000** and suffering an **ABEND0C4 PIC11**, which we see in an IEA995I Symptom Dump message. This is happening under **job TESTJOB**. You would like an **SVC dump**; however, none is being produced.
- SLIP SET,**COMP=0C4,RE=11,JOBNAME=TESTJOB,ADDRESS=(5000),**
  A=SVCD,END
- **NOTES**
  - The **PIC (Program Interrupt Code) on an ABEND0C4** is treated as a **REASON** code by SLIP.
  - The **ADDRESS** parameter is used on a non-PER trap to define the address where the abend occurred or message was issued.

A=SYNCSVCD and A=SVCD default to a match limit (ML) of 1, meaning the SLIP will disable after 1 match.

We don't show the SDATA option on this SLIP. SDATA is used to specify what general areas of storage are to be included in the dump. If SDATA is not specified, SLIP A=SVCD or SLIP A=SYNCSVCD will use a default list of: ALLPSA, CSA, LPA, NUC, RGN, SQA, SUMDUMP, and TRT. (Note: This list is slightly different for A=STDUMP or A=TRDUMP.) I personally like to use the following list: SDATA=(RGN,CSA,LPA,SQA,ALLNUC,TRT,SUM,ALLPSA,GRSQ).

Case 2

Key points:

Using MSGID to qualify error

Using A=TRACE

Including STD,REGS on TRDATA

Using indirection to GTF trace specific data

Case 2

- You have a program that runs frequently, issuing a message with **message ID STRT1** when it starts, and **STOP2** when it ends. Your colleague thinks your program is overlaying a fullword of data in the system **CVT at offset +50**. Write a pair of SLIPs to defend your program's honor by **GTF tracing** this CVT data on entry and exit to your program. (Note: the CVT is pointed to by low core location 10.)

- SLIP SET,**MSGID=STRT1,**
  A=TRACE,TRDATA=(STD,REGS,10?+50,10?+53),END

  SLIP SET,**MSGID=STOP2,**
  A=TRACE,TRD=(STD,REGS,10?+50,10?+53),END

- **NOTES**
  - When using **A=TRACE**, always include **TRDATA=(STD,REGS)** for valuable debugging information. Note that **TRDATA** can be abbreviated as **TRD**.
  - In order for SLIP to write GTF trace records, GTF must be started with the SLIP option.
  - The **indirection** could have been coded in "shorthand" as: **10?+50,+53**

If these SLIPs are set specifying A=TRACE but GTF tracing is not active or has not been started with the SLIP option specified, the SLIPs will still be accepted and enabled. SLIP will put out a warning message that GTF is not active. These SLIPs can still match, even if the action cannot be taken.

Sometimes a debugger will deliberately leave GTF off if they have coded something such as: A=TRDUMP,ML=5 because they want a dump on the 5th occurrence of something but don't really care about the first 4 instances.

Case 3

Key points:

Using PER SA to trap an overlay

Using indirection in the RANGE parameter

Using A=SYNCSVCD on a PER trap

Case 3

- **A real customer case:**

  The product FMID information in the CVTPRODI field of the CVT is being overlaid, with surprisingly devastating system effect. This 8-byte field is in the CVT prefix at offset –X'20'. This field content is established at IPL and should never get changed. Take a dump when this field gets altered. (Note: Low core location X'10' points to the CVT. The CVT lives in common storage.)

- SLIP SET,SA,ASIDSA=SA,RA=(10?-20,10?-19),
  A=SYNCSVCD,END

- **NOTES**
  - **ASIDSA=SA** is used when monitoring alteration of common storage.
  - The indirection in the **RANGE** is resolved when the SLIP is enabled.
  - We could have written the range as: **RA=(10?-20,-19).**
  - **See APAR OA41190!**

A=SYNCSVCD and A=SVCD default to a match limit (ML) of 1, meaning the SLIP will disable after 1 match.

Case 4

Key points:

Using REFBEFOR to correct corruption

Using REFBEFOR to copy corrupted data for diagnostic use

Case 4

- Consider Case 3. If the field is static, we should be able to refresh it, thus preventing the system impact.
  Refresh the field back to its original content of 'HBB7780 ', before taking a dump.

- SLIP SET,SA,ASIDSA=SA,RA=(10?-20,10?-19),
  A=(REFBEFOR,SYNCSVCD),
  REFBEFOR=(10?-20,EQ,C8C2C2F7_F7F8F040),END

- **NOTES**
  - An additional **REFBEFOR** triplet could be added prior to the existing one if we want to **copy the corruption,** thereby preserving it for diagnostic purposes, prior to refresh:
    - **REFBEFOR=(targetaddr,EQC(8),10?-20,…)**
  - Use REFAFTER in the case of an overlay where the damage does not need to be immediately corrected, so that you can see the content of the overlay in the dump.
  - SLIP will update storage a max of 8 bytes at a time.
  - The underscore within the **REFBEFOR** source value is optional and strictly cosmetic.

REFBEFOR and REFAFTER are extremely powerful. We recommend using this option under the guidance of an experienced support representative.

Case 5

Key points:

Using a direct address in the RANGE parameter

Ramifications of using or omitting JOBNAME

Using DATA comparison to filter

Case 5

- You have an application named BADLUCK which runs in its own address space of the same name. The eyecatcher of the application's "WXYZ" control block keeps getting overlaid. This control block lives in common storage at address 1E123000, and its eyecatcher is at offset +0. Take a dump when the storage gets corrupted.

- SLIP SET,SA,ASIDSA=SA,RA=(1E123000,1E123003),
  DATA=(1E123000,NE,E6E7E8E9),A=SYNCSVCD,END

- NOTES
  - **ASIDSA=SA** is used when monitoring alteration of common storage.
  - We don't want to add JOBNAME to the SLIP as that would filter out an overlay done by another job.
  - The **DATA** compare will make sure that we don't match on the case where job BADLUCK is initializing the control block eyecatcher.

Case 6

Key points:

Trapping a Private storage overlay

Using JOBNAME with MODE=HOME

Ramifications of JOBNAME=,MODE=HOME

Case 6

- You have an application named BADLUCK which runs in its own address space of the same name. The eyecatcher of the application's "ABCD" control block is being overlaid. You believe the application itself is responsible for the overlay. The overlaid control block always lives at private storage address 6000, and the eyecatcher of "ABCD" is at offset +0. Take a dump when the storage gets corrupted.

- SLIP SET,SA,ASIDSA='BADLUCK',RA=(6000,6003),
  JOBNAME=BADLUCK,MODE=HOME,
  DATA=(6000,NE,C1C2C3C4),A=SYNCSVCD,END

- NOTES
  - **MODE=HOME** restricts a match to a non-cross memory environment. If work in JOB BADLUCK PC's to another address space, then corrupts the control block in private storage of job BADLUCK, this SLIP won't catch that.
  - Length of constant determines how many bytes of **DATA** are compared.

Remember that ASIDSA indicates the address space that the private storage being monitored resides in. This is *not* a filter for who actually overlays the storage. JOBNAME or ASID must be used to filter on who is doing the alteration.

Case 7

Key points:

Common place scenario of wanting to monitor storage for alteration after it is GETMAINed

Using dynamic PER (A=TARGETID)

Use of registers in the indirection in the RANGE parameter

Using PVTMOD

Using ID to name a SLIP

The BPER symbolic

Setting SLIPs as disabled originally then enabling desired SLIP

Case 7

- Same as Case 6 except that this time we don't know where the private storage-resident "ABCD" control block lives until after it is GETMAINed. It is GETMAINed by private load module GETSTOR at offset +X'1B0'. On return from a GETMAIN, register 1 contains the address of the GETMAINed storage. Once we have this information, we can set up the SA SLIP.

- ```
  SLIP SET,IF,DISABLE,P=(GETSTOR,1B0),ID=SLP1,
            JOBNAME=BADLUCK,MODE=HOME,
            A=TARGETID,TARGETID=SLP2,END

  SLIP SET,SA,DISABLE,RA=(1R?+0,1R?+3),ASIDSA='BADLUCK',
           DATA=(BPER?+0,NE,C1C2C3C4),ID=SLP2,
           A=SYNCSVCD,END

  SLIP MOD,ENABLE,ID=SLP1
  ```


In this example, we're assuming that GETSTOR+1B0 points to the return point from the GETMAIN.

If you accidentally try to enable both SLIPs simultaneously, SLIP processing will detect that SLP2 is targeted by SLP1 and will leave it disabled.

A dynamic PER trap chain can consist of two or more traps. The chain cannot be circular.

When PVTMOD=(modname,x,y) is specified but the "y" part is omitted, this is equivalent to "y" = "x", that is, the SLIP is targeting the single byte at offset x. The same is true for PVTEP, LPAMOD/LPAEP, and NUCMOD/NUCEP.
  ```

Case 7: Notes

- Notes:
  - When one PER trap targets another as it matches and disables, this is called **dynamic PER.**
  - We have given each SLIP an ID, which can be a maximum of 4 characters.
  - The **RANGE** of the second SLIP is **resolved at the time the SLIP is enabled,** using the value in register 1 at the time the first SLIP matched and disabled.
  - The **DATA** parameter of the second SLIP is **resolved when a PER event occurs** for that SLIP, that is, when the specified range is altered.
  - The symbolic **BPER** can be used to indicate the **B**eginning address of the **PER** range.
  - When using dynamic PER, it is helpful to **set all SLIPs disabled** originally, then **enable the first in the chain.**
  - We do not need to specify JOBNAME and MODE on the second SLIP. **The environmental specifications on the first SLIP automatically apply to all SLIPs in a dynamic PER chain.** This cannot be overridden!

There is one other SLIP symbolic: BEAR . This is set up to be the Breaking Event Address from the Breaking Event Address Register. This is the address of the last instruction on this CP to cause a branch or a change in linear flow of execution (e.g. PC or LPSW) prior to the PER interrupt.

Case 8

Key points:

Trapping a Private storage overlay that could originate from outside the address space

Ramifications of omitting JOBNAME=,MODE=HOME

Qualifying a direct address with a jobname

Case 8

- You have an application named BADLUCK which runs in its own address space of the same name. The eyecatcher of the application's "ABCD" control block is being overlaid. ***This application interfaces with a number of other address spaces, any of which could have done the overlay.*** The overlaid control block always lives at private storage address 6000, and the eyecatcher of "ABCD" is at offset +0. Take a dump when the storage gets corrupted.

- **SLIP SET,SA,RA=(6000,6003),ASIDSA='BADLUCK',**
  DATA=('BADLUCK'.6000,NE,C1C2C3C4),
  A=SYNCSVCD,END

- **NOTES**
  - Since we don't know what address space or cross memory environment the overlayer is running in, ***we have removed JOBNAME=BADLUCK and MODE=HOME.***
  - Since we don't know which address space will be current when the SA occurs, we must qualify the address of the **DATA** compare to indicate that address **6000** is within job **BADLUCK**.

Case 9

Key points:

- Using an Instruction Fetch PER trap
- Using LPAMOD to define the RANGE of a PER trap
- Using registers as part of indirection in DATA parameter

Case 9

- CSECT MYMOD lives at offset X'A00' thru X'AFF' in LPA load module MYLOAD. To debug a logic problem, you want to trace the instruction flow through the entire CSECT, writing a GTF record for each instruction. You want to trace the X'20' byte work area pointed to by Register 11, as well as basic environmental information.

- SLIP SET,IF,L=(MYLOAD,A00,AFF),A=TRACE,
  TRDATA=(STD,REGS,11R?+0,+1F),END

- NOTES
  - For **A=TRACE**, the default is to have no limit on how many times the SLIP can match. Use the ML parameter in you want a match limit on the SLIP.
  - Always specify **STD** and **REGS** to get fundamental diagnostic information.
  - Note that the X'100' byte range that we are trapping is not huge, but if the code executes very frequently or loops, this could cause impact.

Case 10

Key points:

- Common scenario where SLIP is used to trace data on entry to and exit from a routine
- Using A=IGNORE to disregard intervening instructions
- Specifying multiple data pairs in TRDATA
- Using 64-bit register notation
- Using wildcarding to enable a set of SLIPs

Case 10

- The results from the Case 9 SLIP suggest the problem is related to bad parameter list content. Trace the parameter list ONLY on entry to and exit from MYMOD. (Assume the last instruction of MYMOD is offset X'FE' into the CSECT.) The parameter list is pointed to by Register 1, is X'10' bytes long, and lives above the bar. Also trace the last X'20' bytes of the X'30' byte control block pointed to by the second word of the parameter list. This block lives below the bar.

- SLIP SET,IF,D,L=(MYLOAD,A00,AFF),ID=SLP1,A=TRACE,
  TRDATA=(STD,REGS,1G!+0,+F,1G!+4?+10,+2F),END

  SLIP SET,IF,D,L=(MYLOAD,A01,AFD),ID=SLP2,
  A=IGNORE,END

  SLIP MOD,ENABLE,ID=SLP*

Case 10: Notes

- This is not a violation of the rule that you can only monitor one PER range at a time.
  - Software parsing of SLIP syntax detects:
    - The **SLP2 range** is a subset of the **SLP1 range**
    - SLP2 has **A=IGNORE**
  - **PER CR10 and CR11 will hold the range from SLP1**
  - Software filtering will determine whether a PER event has occurred within the subset **range** defined by SLP2 and take action accordingly.
    - PER EVENT ignored for all but the first and last instruction
    - First and last instruction produce trace data
- The order of SLIP entry is important! SLIP software processes traps in a LIFO order, and we want it to encounter the A=IGNORE trap first.
- Enter both SLIPs disabled with similar IDs, use wildcarding to enable simultaneously.
- In TRDATA we must use "xG!" instead of "xR?" to perform 64-bit interpretation of the register.

A=IGNORE defaults to no match limit.

Note that an interrupt will occur on every instruction in this PER range, even though we are IGNORE-ing all but the first and last instructions!

Case 11

Key points:

Using SLIP A=SUBTRAP to take different actions in different subsets of the range being monitored

Case 11

- What if we want to combine our actions from cases 9 and 10? Let's trace the parameter list on our first and last entries, let's trace the X'20' byte work area pointed to by Register 11 for all the in between entries, and let's take a dump and stop the GTF trace if, on exit from MYMOD, we have a non-zero value in Reg15!

- SLIP SET,IF,D,L=(MYLOAD,A00,AFF),ID=SLP1,A=TRACE,
  TRDATA=(STD,REGS,1G!+0,+F),END

  SLIP SET,IF,D,L=(MYLOAD,A01,AFD),ID=SLP2,
  A=(SUBTRAP,TRACE),TRDATA=(STD,REGS,11R?+0,+1F),END

  SLIP SET,IF,D,L=(MYLOAD,AFE),ID=SLP3,
  A=(SUBTRAP,SYNCSVCD,STOPGTF),DATA=(15R,NE,0),END

  SLIP MOD,ENABLE,ID=SLP*

Case 11: Notes

- This is not a violation of the rule that you can only monitor one PER range at a time.
  - Software parsing of SLIP syntax detects:
    - The SLP2 and SLP3 RANGEs are subsets of the SLP1 RANGE
    - SLP2 and SLP3 have **A=SUBTRAP**
  - **PER CR10 and CR11 will hold the RANGE from SLP1.**
  - Software filtering will determine whether a PER event has occurred within the subset RANGEs defined by SLP2 or SLP3 and take action accordingly.
- Once again the order of SLIP entry is important!
- Once again we enter the SLIP set disabled, then use wildcarding to enable all simultaneously.

A=IGNORE defaults to no match limit.

Note that an interrupt will occur on every instruction in this PER range, even though we are IGNORE-ing all but the first and last instructions!

Case 12

Key points:

Using a Successful Branch PER trap

Using NUCMOD (N) to define RANGE on a PER trap

Using NUCMOD with no offsets specified

Using A=STRACE,STDATA=

Using MATCHLIM

Case 12 (a breather!)

- You are suffering an abend in your nucleus module SVC201. An SVC dump is being produced by your recovery. However, in order to better understand the code flow leading up to the abend, you would like to see all branches within SVC201 in the system trace table. Include the X'10' byte area pointed to by Register 6 in the trace data.

- SLIP SET,SBT,N=(SVC201),
  A=STRACE,STDATA=(6R?,+F),ML=10000,END

- **NOTES**
  - **A=STRACE** causes an SPER system trace entry to be written.
  - A maximum of X'14' bytes of data may be written in an SPER entry.
  - Default **MatchLim** for **A=STRACE** varies depending on SLIP parameters specified. For this example, the default is 50, so we code our own to be much larger.
  - When no starting/ending offset is specified on **NUCMOD** (or LPAMOD or PVTMOD), the entire module is monitored.

Case 13

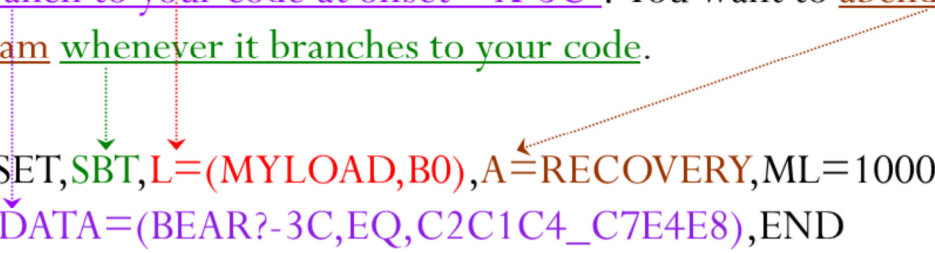
Key points:

Using A=RECOVERY

Using BEAR symbolic

Case 13

- A rogue program keeps branching into your code at offset +B0 in LPA module MYLOAD. The rogue program moves around in storage, but it has the eyecatcher 'BADGUY' at offset +0 and makes the branch to your code at offset +X'3C'. You want to abend this program whenever it branches to your code.



- SLIP SET,SBT,L=(MYLOAD,B0),A=RECOVERY,ML=1000,
  DATA=(BEAR?-3C,EQ,C2C1C4_C7E4E8),END

- **NOTES**
  - A=RECOVERY results in the interrupted unit of work being targeted with an ABEND06F.

Any questions??