Session 10867
z/OS Debugging: *Old Dogs and New Tricks*

MVS Core Technologies Project – March 13th 2012

Patty Little Jerry Ng
IBM Poughkeepsie
plittle@us.ibm.com jerryng@us.ibm.com

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

The following are trademarks or registered trademarks of other companies.

Java and all Java-related trademarks and logos are trademarks of Sun Microsystems, Inc., in the United States and other countries.
Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.
Microsoft, Windows and Windows NT are registered trademarks of Microsoft Corporation.
UNIX is a registered trademark of The Open Group in the United States and other countries.
SET and Secure Electronic Transaction are trademarks owned by SET Secure Electronic Transaction LLC.

* All other products may be trademarks or registered trademarks of their respective companies.

Notes:

Performance is in Internal Throughput Rate (ITR) ratio based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput improvements equivalent to the performance ratios stated here.

IBM hardware products are manufactured from new parts, or new and serviceable used parts. Regardless, our warranty terms apply.

All customer examples cited or described in this presentation are presented as illustrations of the manner in which some customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics will vary depending on individual customer configurations and conditions.

This publication was produced in the United States. IBM may not offer the products, services or features discussed in this document in other countries, and the information may be subject to change without notice. Consult your local IBM business contact for information on the product or services available in your area.

All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.

Information about non-IBM products is obtained from the manufacturers of those products or their published announcements. IBM has not tested those products and cannot confirm the performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Prices subject to change without notice. Contact your IBM representative or Business Partner for the most current pricing in your geography.

Agenda

- Debugging with the PSW
- SYSTRACE enhancements for identifying CPU time usage
- Using BEAR for problem diagnosis

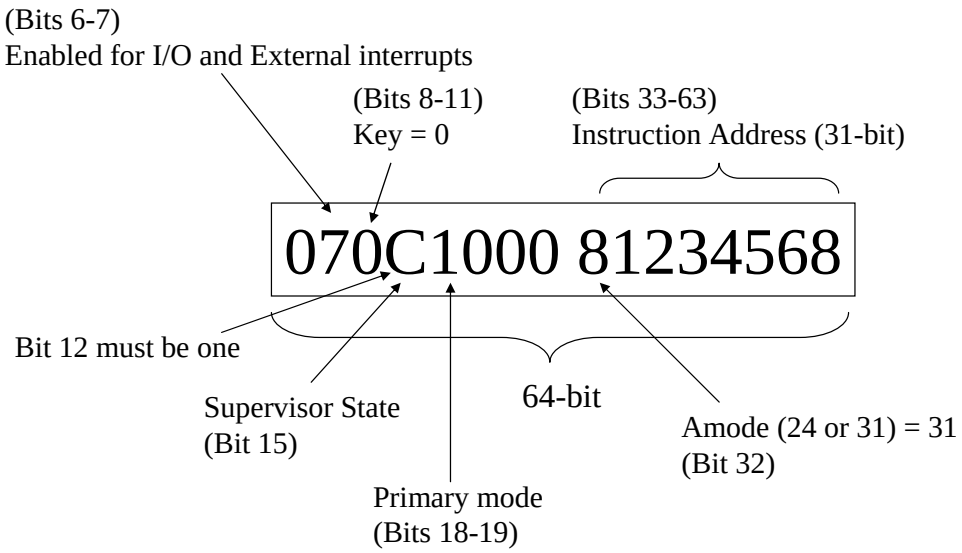
This presentation will discuss some of the old debugging concepts, as well as new techniques introduced by the recent z/OS releases.

Debugging with the PSW

- **Together with the registers, it is the fundamental way of problem diagnosis**
- **Gives you clues about the problem:**
  - The environment at the time of error
  - The module related to the problem
  - The reason for the failure
  - Other related information

The PSW and registers at the time of error provides good information that can help you to solve the problem. They should never be ignored.

A long time ago - ESA/390 mode PSW



The ESA/390 mode PSW was introduced many years ago. It is explained in the Principles of Operations manual. This slide mentioned a few interesting bits in the PSW.

The first bit on the left is bit 0 or the high order bit. Bits 6 and 7 are masks indicating whether I/O and external interrupts are enabled. In this example these bits are on which indicate that these interrupts are enabled. If these bits are off, the program is running with interrupts disabled. Disablement is a form of serialization to protect CPU related resources. A program cannot perform certain functions, such as issuing SVCs, when running disabled.

Bits 8-11 represent the key of the PSW. This key must match the storage key when write to storage is attempted, unless the PSW key is zero.

Bit 12 must be one, or else a specification exception (PIC 6) will occur. Bits 18 to 19 are the ASC mode bits. Bit 32 is the AMODE bit.

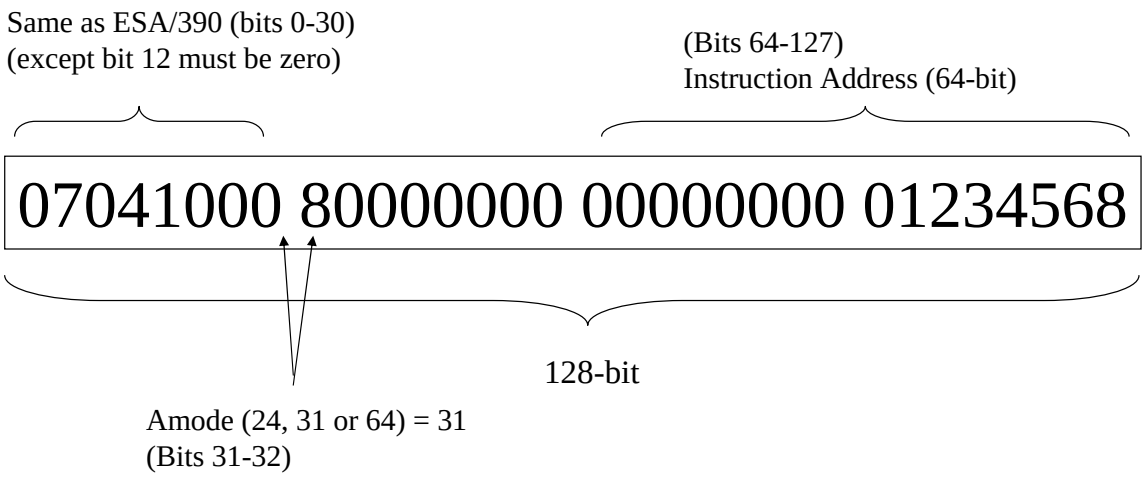
Example: debugging an ABEND0C4

```
Time of Error Information

  PSW: 077D0000 80C19D6E   Instruction length: 04   Interrupt code: 0004
  Failing instruction text: 1BFF5040 10304 7FF 0008B20A

  Registers 0-7
  GR: 00039C80 00F425D0 C00425D0 00C9B000  FFFFFFFF 00000000 00000000 00000000
  AR: 00000A38 00000000 00000000 00000000  00000000 00000000 00000000 00000000
  Registers 8-15
  GR: FFFFFFFF 00000000 00000000 007CAD84  007CAA38 007C7F50 80FD5288 00000000
  AR: 00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
```

- **ABEND0C4 is due to PIC 4 (protection exception)**
- **Failing instruction is 50401030 (STORE using R1)**
- **PSW contains key of 7**
- **Key of storage at F425D0 should be investigated**
  - For a STORE, if PSW key does not match storage key PIC 4 will occur (unless PSW key is 0)

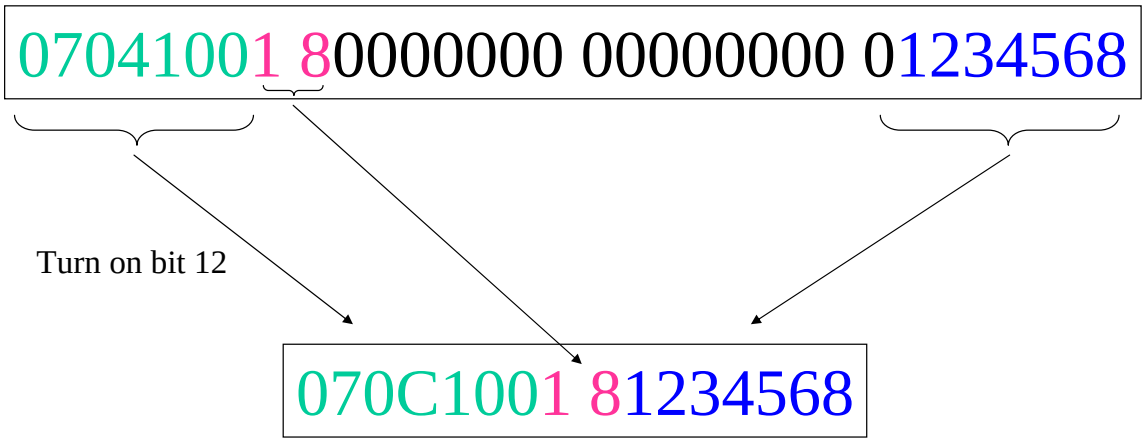
This is an example to demonstrate the use of a PSW in debugging. The error is a PIC 4 (Program Interrupt Code 4 = Protection Exception). The PSW points to the middle of the failing instruction text. Since the instruction length is 4, backing up 4 bytes shows that the failing instruction is a store using register 1. The PSW contains a key of 7. For a store, the PSW key must match the key of the storage, unless the PSW key is zero. The key of the storage in register 1 should be investigated.

Some time ago - z/Architecture mode PSW



The z/Architecture mode PSW was introduced with the 64-bit architecture. Since the instruction address can now be 64-bit, the PSW is double the size of the ESA/390 PSW. The first word of this PSW is quite similar to the 390 PSW, except that bit 12 must be 0, and bits 31 and 32 indicate the AMODE. The second word of this PSW is mostly zero.

PSW Scrunching: 128-bit to 64-bit



PSW scrunching is performed by z/OS system modules
This is possible if the PSW instruction address resides below the bar

Prior to z/OS V1 R13, program execution above the 2G bar is not supported. This means the instruction address should always be 31-bit. So the second and third word of the PSW are mostly zeros (except bit 32). z/OS takes advantage of this condition and scrunches the 128-bit PSW into 64-bit before saving it in many places.

IPCS ST FAILDATA & VERBX LOGDATA

```
Time of Error Information

  PSW: 077C2001 9FAEFF86   Instruction length: 04   Interrupt code: 0038
  Failing instruction text: 0033A784 00389180 3013A784
```

```
Time of Error Information

   PSW: 04046000 80000000 00000000 0178F356
   Instruction length: 04   Interrupt code: 0004
   Failing instruction text: 000A5023 00005032 00044172
```

- **Time of Error Information can have scrunched or 128-bit PSW**
  - depending on the recovery routine and SDWA
- **For TCB mode errors, you can find the 128-bit PSW in the RTM2WA**

From a dump, you will usually find the 128-bit PSW in the output of IPCS ST FAILDATA or VERBX LOGDATA. But sometimes you may see the scrunched 64-bit PSW. This is most likely due to a very old recovery routine that did not request an above-the-line SDWA. For TCB mode errors, you can always find the 128-bit PSW in the RTM2WA.

Finding the 128-bit PSW in RTM2WA

From IPCS ST FAILDATA

```
Time of Error Information

  PSW: 070C0001 8AF8D180   Instruction length: 04   Interrupt code: 010
  Failing instruction text: 17885810 10205010 301847F0
```

From IPCS SUMM FORMAT ASID(n)

```
RTM2WA: 7FFAFE10
   +0000  ID....... RTM2      ADDR..... 7FFAFE10  SPID..... FF        LGTH..... 0011F0
   +0014  VRBC..... 009FD550  ASC...... 00F882A0  CCF...... 84        CC....... 0C4000
.....
....   Lines omitted here
....
   +06C8  TRNE..... 00000000  072FF800
   +06D0  BEA...... 00000000  0AF8B552
   +06D8  PSW1..... 07040001  80000000  00000000  0AF8D180
```


This example shows a scrunched PSW in the IPCS ST FAILDATA output. If this error occurred in TCB mode, one can find the 128-bit PSW in the RTM2WA. This control block is formatted after the failing TCB in the IPCS SUMMARY FORMAT output.

IPCS SYSTRACE

In z/OS R12 and lower

```
01 010D 006F0858  DSP         070C2000 BF5125F0  00000000 BF6C275A 0009B968
01 010D 006F0858  SVC       2 078C0000 BF6968A0  3F696844 00000000 3F781288
01 010D 006F0858  PGM     004 078C1000 81452E60  00040004 00000000
                                                           00000000
01 01BD 006E0758 *RCVY  PROG                     940C4000 00000004 00000000
```

In z/OS R13

```
01 00E4 009FF800  DSP         00000000_0AF2F598  00000000 E5000800 72F65720
                              07040000 80000000
01 00E4 009FF800  PGM     010 00000000_0AF8D180  00040010 00000008
                              07040001 80000000            072FF800
01 00E4 009FF800 *RCVY  PROG                     940C4000 00000010 00000000
```

- **z/OS R13 supports program execution above the bar, as long as that program does not invoke system services**
  - Some system trace records now contains 128-bit PSWs
  - Instruction address is displayed on the first line, followed by the first half of the PSW on the next line

The system trace table consists of many different kinds of entries. Prior to z/OS V1 R13, all PSWs in the system trace entries are 64-bit (scrunched). z/OS V1 R13 is the first release that will support program execution above the 2G bar, as long as the program does not invoke system services. Since the instruction address can now be greater than 31 bit, some of the system trace entries have been changed to contain 128-bit PSWs. Note that the 128-bit PSW is displayed in 2 lines, but not in the order of bit 0 to bit 127. The instruction address is displayed first, then followed by the first half of the PSW in the next line.

RB in IPCS SUMM FORMAT ASID(n)

In z/OS R13

```
SVRB: 009FD548
   -0020  XSB...... 7FFF9338  FLAGS2... 00        RTPSW1... 070C0001
   -0014            89F8D186  RTPSW2... 0006003B  7FFFFBD0
   -0008  FLAGS1... 02000000  WLIC..... 0006003B
   +0000  RSV...... 00000000  00000000            SZSTAB... 001ED022
   +000C  CDE...... 00000000  OPSW..... 070C0001  89F8D186
```

```
XSB: 7FFF9338
   +0000  XSB...... XSB       LINK..... 00000000  XLIDR.... 00000000
   +0014  XLAS..... 00000000  TKN...... 0000      ASD...... 0000
......
....   Lines omitted here
...
+00D4  AX....... 0000      PASID.... 00D4
+00D8  BEA...... 00000000  09F8BF52
+00E0  PSW16.... 07040001  80000000  00000000  09F8D186
```

- RBOPSW still exists but it is not relied upon
- XSBOPSW16 contains the official 128-bit PSW

The RB contains a field called RBOPSW, which is used by z/OS to save the interrupted PSW of a TCB. This PSW is 64-bit and will still exist in z/OS V1 R13. It will be maintained by z/OS, but it will not be not used by z/OS to re-dispatch the TCB. The official 128-bit PSW is now in the XSB. In a dump, you should usually find that the RBOPSW and XSBOPSW16 are similar.

Debugging with the PSW - Summary

- **PSW and registers are the signature of a program**
- **Use them to explain the error in the dump**
- **Use the 128-bit PSW (if available) for debugging, especially for:**
  - Program checks
  - Problems from programs running in AMODE 64

The PSW and registers at the time of error provides good information that can help you to solve the problem. They should never be ignored. Find the 128-bit failing PSW when you are debugging a problem, especially if it is related to a program check or a module running in AMODE 64.

SYSTRACE PERFDATA

- **Introduced at z/OS R12**
- **Provides a "performance breakdown" on system trace data**
  - SRB, TCB, and total CPU time used per address space in trace
  - Breakdown of SRB usage by address space
  - Breakdown of TCB usage by address space
  - I/O times
- **Caveats**
  - **Only tells the story from what events get traced in system trace**
  - **Only tells the story from the timeframe of system trace (short!)**
  - **Data must be interpreted! e.g. DB2 may naturally have large totals**
  - **Does not take the place of other performance tools**

The SYSTRACE PERFDATA option is a valuable addition to the IPCS diagnostic toolkit when properly applied; however, it can be dangerous when misused. The statistical data provided by PERFDATA must be applied within the context of the problem. For example, if the external symptom is high CPU utilization in Master, but SYSTRACE PERFDATA shows DB2 as being the predominant user of CP, then further investigation is required to determine which of a number of possibilities exist: 1) the dump does not accurately reflect the problem (bad timing – remember that a system trace snapshot is typically very short), 2) DB2 is normally a heavy CP user so the numbers in PERFDATA are normal, or 3) there is a connection which still needs to be identified between the external symptom of Master CPU usage and the internal observation of high DB2 CPU.

SYSTRACE PERFDATA is documented in the z/OS MVS IPCS Commands manual.

SYSTRACE PERFDATA: CPU Summary

- **Command: IPCS SYSTRACE PERFDATA**
- **For each CP, shows time range of events that were analyzed**
  - Only reports on ranges where all CPs are represented
    - Between "TRACE DATA IS NOT AVAILABLE FROM ALL PROCESSORS BEFORE THIS TIME" and "TRACE DATA IS NOT AVAILABLE FROM ALL PROCESSORS AFTER THIS TIME"
    - Starts/Ends range with a dispatch entry: DSP/SRB/SSRB
    - Idle time: time in processor no work WAIT
    - CPU Overhead: time not attributable to SRB time, TCB time, or Idle time

CPU#	Went from	To	Seconds	SRB Time	TCB Time	Idle Time	CPU Overhead
00	16:02:24.880423	16:02:28.200541	3.320117	1.370858	1.943048	0.000000	0.040452
01	16:02:24.880437	16:02:26.878804	1.998367	1.371296	0.621700	0.000000	0.036751
			5.318485	2.742155	2.564749	0.000000	0.077204

To understand why SYSTRACE output has sections where not all CPs are represented, it is necessary to understand how the operating system manages its trace buffers. There is one trace buffer associated with each CPU. Trace buffers are finite in size and so will "wrap-around" when they fill up. Trace entries are constantly being added and causing the oldest entries in the buffer to be displaced. A CP that is executing work that is causing lots of trace entries to be written will wrap its trace buffer quite quickly. This means that the time spanned by the entries in the trace buffer is quite short. A CP that is executing work that is producing very few entries will take longer to wrap its trace buffer. This means that the time spanned by the entries in the trace buffer will be longer. The SYSTRACE formatter merges the trace entries by time stamp. Therefore, the oldest entries from the CPUs with slower-filling buffers will appear first in the trace. Eventually trace processing reaches a timeframe where even the most rapidly filling buffers have entries. At this point "trace data is available from all processors".

SYSTRACE PERFDATA: Address Space View

- **For each address space in designated trace table time range:**
  - Total SRB time
  - Total TCB time
  - Total time
- **Formatted in order that ASIDs are encountered in trace**

```
Found 53 address spaces in SYSTRACE.
Found 145 SRB and SSRB PSWs in SYSTRACE.

CPU breakdown by ASID:

ASID Jobname    SRB Time     TCB Time    Total Time
---- -------- ------------ ------------ ------------
006B ABCDEFGH     0.001601     0.012151     0.013752
0060 CICSRGNA     0.005703     0.202266     0.207969
0078 SOMEBAT1     0.000994     0.118045     0.119040
0080 SOMEBAT6     0.006793     0.057610     0.064404
0064 PBLTEST      0.008914     0.045282     0.054196
0085 CICSRGNB     0.000725     0.074175     0.074901
0063 MYJOB        0.001004     0.112320     0.113325
          . . . . .
```


This address space-related CPU usage data can be sorted by SRB time, or TCB time, or Total time by using the Report View option under IPCS. To use this, first go to the bottom of the report. Then enter REPORT VIEW on the command line. This will put you into an editor which will allow you to delete, exclude, “find all”, and sort lines. Typing COLS over the line number will give you a column indicator to aid in sorting.

SYSTRACE PERFDATA: Address Space by SRB View

- For each dispatched SRB for each address space in designated trace table time range:
  - Individual and Total SRB CPU times
  - SRB dispatch PSW and Frequency of occurrence
- Formatted in order that ASIDs are encountered in trace

SRB breakdown by ASID:

ASID	Jobname	SRB PSW	# of SRBs	Time
006B	ABCDEFGH	070C0000 81174100	97	0.001601

ASID	Jobname	SRB PSW	# of SRBs	Time
0060	CICSRGNA	070C0000 813C1348	273	0.003070
0060	CICSRGNA	070C0000 8102E876	20	0.000046
0060	CICSRGNA	070C0000 886D0656	7	0.000058
0060	CICSRGNA	070C0000 91C962E8	59	0.000688
0060	CICSRGNA	070C0000 81174100	140	0.001662
0060	CICSRGNA	070C0000 928EE768	13	0.000141
0060	CICSRGNA	070C3000 80FF2768	2	0.000006
0060	CICSRGNA	070C3000 813C3F02	5	0.000026
0060	CICSRGNA	070C0000 816A88AC	1	0.000002
				0.005703

Once again, users of PERFDATA are cautioned to use judgment in their interpretation of the data. Having many SRBs dispatched at the exact same PSW address may be perfectly normal for a particular function, especially if the total CPU time of the SRB activity dispatched at this point is quite low. PERFDATA must always be applied in context with other debugging symptoms and clues.

SYSTRACE PERFDATA: Address Space by TCB View

- For each dispatched TCB for each address space in designated trace table time range:
  - Individual and Total TCB CPU times
  - Number of dispatches of each TCB
- Formatted in order that ASIDs are encountered in trace

```
TCB breakdown by ASID:

ASID Jobname  TCB Adr  # of DSPs     Time
---- -------- -------- --------- ------------
006B ABCDEFGH 009EB748        97     0.012151


ASID Jobname  TCB Adr  # of DSPs     Time
---- -------- -------- --------- ------------
0060 CICSRGNA 009FA4E0       733     0.201798
0060 CICSRGNA 009C0E88        20     0.000319
0060 CICSRGNA 009FAB20        22     0.000148
                                  ------------
                                     0.202266
```


The same comment made in the speaker notes of the previous slide applies to this slide as well. Large counts of dispatches may be normal, or could be a sign of a problem. It is important to map this data against other symptoms that you know about the problem; it cannot be used in a vacuum.

SYSTRACE PERFDATA: SSCH to I/O View

- **For each device:**
  - SSCH time, I/O time, and elapsed time
  - Summary: fastest, slowest, average, and total time

SSCH to I/O times:

Device	SSCH Issued	I/O Occurred	Duration
2080	16:02:24.880476	16:02:24.880680	0.000203
2080	16:02:24.881121	16:02:24.881347	0.000225
2080	16:02:24.882362	16:02:24.882596	0.000233
2080	16:02:24.882970	16:02:24.883410	0.000440
2080	16:02:24.883555	16:02:24.883907	0.000352
2080	16:02:24.884095	16:02:24.884355	0.000260
2080	16:02:25.499662	16:02:25.499891	0.000228
2080	16:02:25.499967	16:02:25.500265	0.000298
2080	16:02:25.501723	16:02:25.501953	0.000230
2080	16:02:25.502092	16:02:25.502336	0.000243
2080	16:02:25.502648	16:02:25.502856	0.000208
2080	16:02:25.503104	16:02:25.503326	0.000221
2080	16:02:25.503398	16:02:25.503603	0.000204
2080	16:02:25.503723	16:02:25.503951	0.000227
2080	16:02:25.531892	16:02:25.532116	0.000224
2080	16:02:25.532181	16:02:25.532423	0.000241
			0.004044

```
Events for 2080 :            16
   Quickest I/O    :     0.000203
   Slowest  I/O    :     0.000203
   Total           :     0.004044
   Average         :     0.000252
```


This portion of the report may be useful in identifying slow I/O response time. It is sorted by device number. Data reported for each device is unsorted. Use REPORT VIEW if you would like to sort it. Note that the report does provide quickest, slowest, total and average I/O times for each device immediately after the list of “SSCH to I/O times”.

SYSTRACE PERFDATA: "Locks and Clocks" View

- Itemizes CMS lock suspensions
- Displays PSWs on CLKCs entries, sorted by PSW address within ASID
  - Aids in loop identification
  - Use DOWHERE parameter to map PSW addresses to a module:
    SYSTRACE PERFDATA(DOWHERE)

```
Lock Events:
Lock ASID TCB/SRB  Type PSW Adr   Suspended at     Resumed at    Suspend Time
---- ---- -------- ---- -------- --------------- --------------- ------------
CEDQ 0009 009F6638 TCB  9276C4D8 16:02:25.532319 16:02:25.532422     0.000102
CEDQ 0001 0099D0E8 TCB  9276C4D8 16:02:25.532572 16:02:25.532610     0.000037
                                                                   ------------
                                                    2 suspends     0.0000

CLKC Events:

ASID Jobname  SRB/TCB      Clkc PSW      Where processing (CPU usage for this ASID is:     0.113325)
---- -------- -------- ---------------- ------------------------------------------------------------------
0063 MYJOB    009EB968 078D1E00 800235D4 ASID(X'0063') 000235D4. AREA(Jobpmyjob)+0215D4 IN PRIVATE
0063 MYJOB    00000000 070C2000 813C13AE ASID(X'0063') 013C13AE. IEANUC01.ICYPGAD+66 IN READ ONLY NUCLEUS
0063 MYJOB    009EB968 078D0E00 84BFE78C ASID(X'0063') 04BFE78C. IDA019L1+278C IN EXTENDED PLPA
```


The lock statistics reported may be useful if assessing whether there are many significant delays due to CMS lock contention.

Notice how the DOWHERE option on PERFDATA breaks the CLKC PSWs down into load module and offset. This is convenient if you have already identified a loop in the system trace table (many CLKC entries in a row in a system trace table with similar PSWs characterizes an enabled loop) and you want a short cut to mapping the PSW addresses.

Spin Locks and CPU Usage

- **z/OS uses spin locks to serialize various functions**
  - Many components use spin locks: RSM, IOS, VSM, etc
- **When a unit of work obtains a spin lock, it is automatically disabled so that it cannot be interrupted**
  - Disabled code does not typically write system trace entries
- **If a unit of work "B" wants a spin lock that is held by another unit of work "A", then "B" is placed into a system-monitored spin until the lock is available**
  - The unit of work holding the lock (A) is using CPU
  - The unit of work spinning for the lock (B) is using CPU
  - Spinning for 40 seconds* or more triggers system spin loop actions
  - Spinning for less than 40 seconds was invisible … until now!

*** Default spin time**

A "spin lock" is a serialization mechanism used by many system services. There are many types of spin locks since there are many types of system services that need to be serialized. Sometimes significant contention arises between users of a spin lock, resulting in impact to system performance. The operating system automatically detects spins of length 40 seconds or greater. However, lock contention that results in shorter spins can also cause operating system impact, yet be much more difficult to discover.

Mind the Gap: SYSTRACE SPIN entries

- **Introduced in z/OS R11**
- **Indicates "extended" period of time spent spinning for a spin lock**
  - Trace SPIN LKX/S "start" entry if spinning for >1 second
  - Trace SPIN LKX/P "stop" entry when lock acquired
- **I'm an application, why do I care?**
  - Applications use system services that use spin locks
  - System service performance impacts application performance
- **Other SPIN record types exist besides LKX (spin lock)**
  - See MVS Diagnosis: Tools and Service Aids for complete list

Many system services are serialized by spin locks. While small amounts of contention between system services for a spin lock are completely normal, large amounts of contention can result in slow performance by the system services requiring the lock. This in turn can affect the performance of applications that use that system service.

Spins for a lock that last less than a second do not result in a SPIN record being written.

The type of SPIN record that we will be focusing on is the spin for a lock. The LKX eyecatcher in the trace record is actually referring to the module IEAVELKX that handles spins for locks. Other types of SPIN records exist, and generally have an identifier of format "SPIN zzz/S" or "SPIN zzz/P, where zzz is a 3 letter eyecatcher related to the system module doing the spinning.

Sample SPIN LKX entry

- SPIN LKX/S entry written after spinning for lock for 1 second
- SPIN LKX/P entry written at end of spin

- Unique 1: First byte is approximate length of spin in seconds. (Remaining 3 bytes represent fraction of second.)
- Unique 2: Content of lockword being spun for; identifies CPU that unit of work holding lock is active on. ("AND" off the "80" bit in byte 3)
- Unique 3: Lockword address
- Unique 4: Lock being requested (see PSACLHS in MVS Data Areas)

```
PR ASID WU-ADDR- IDENT  CD/D PSW----- ADDRESS-  UNIQUE-1 UNIQUE-2 UNIQUE-3  PSACLHS- PSALOCAL PASD SASD TIMESTAMP-RECORD
                                                UNIQUE-4 UNIQUE-5 UNIQUE-6  PSACLHSE
01 0005 02391380  SPIN LKX/S          8174C4B8  01000000 00000080 021C3C04  88000000 00000000 0003 0005 C8EC639E6298C7A0
                                                00004000 000002F8 00FF15C2  00000000

01 0005 02391380  SPIN LKX/P          8174C4B8  01482AA2 00000080 021C3C04  88000000 00000000 0003 0005 C8EC639EAAC36480
                                                00004000 000002F8 00FF15C2  00000000
```

About 1 second into spin

Spun for about 1.25 seconds

RSM Address Space Lock

Lock held by work on **CP 0**

From the presence of these records in the system trace table, we can recognize that there was a spin for a system lock that lasted for longer than 1 second. The SPIN LKX/P (STOP) record indicates the length of the spin in the Unique-1 field. The first byte is approximately the number of seconds of the spin, and the remaining 3 bytes represent a fraction of a second. For example, in this STOP record, we see Unique-1 = 01482AA2. This tells us the spin lasted approximately 1sec + X'48'/X'100' which is a little more than 1.25 seconds. From either record we can identify the lock that is being requested by referring to the Unique-4 field. This word of data will have one bit on identifying the lock being requested. This bit string can be mapped against the 4-byte PSACLHS field which is described under the PSA control block mapping in MVS Data Areas. The Unique-2 field provides the indication of who is holding the lock that this unit of work is spinning for. This field contains a logical processor ID, so the unit of work currently active on that processor is the one holding the lock. To convert a logical CP ID to the CPU number that you will see in the PR column of the system trace table, simply turn off the X'00000080' bit. In the example above, when the X'00000080' bit is turned off, the result is all zeros, which tells us that the unit of work holding the lock is on CP 0. We would want to look at that CP next to understand what system service it may have been running in.

Is a System Trace SPIN entry a problem?

- **Spin processing happens regularly on the operating system**
- **The operating system does not get excited about short spins and you are encouraged to follow its example! ☺**
  - Some requests for system services make significant requirements of that service, e.g. RSM High Virtual storage requests
  - Bottom line: big requests take time, perhaps several seconds
- **If you see a SPIN record in the trace while investigating a performance problem, consider it a possible clue**
- **Otherwise, consider it normal processing**

An occasional SPIN record in a system trace table in the absence of performance issues is not something to be concerned about. There are many reasons the operating system may experience a short spin. However, very long spins or more-than-occasional shorter spins, particularly in the presence of a system performance problem, bear further investigation.

A Real Life Example: Using SYSTRACE PERFDATA

- Reported problem: system was intermittently "sluggish"
- Documentation: Console dump (SVC dump) during period of system sluggishness
- **SYSTRACE PERFDATA** showed the following data which the debugger has sorted in increasing "Total Time" order using the "Report View" option under IPCS:

```
ASID Jobname    SRB Time     TCB Time    Total Time
---- -------- ------------ ------------ ------------
001C VLF          0.000005     0.000016     0.000021
0003 RASP         0.000053     0.000000     0.000053
0026 PBLTEST2     0.000054     0.000053     0.000108
-  -  -  45 Line(s) not Displayed -  -  -  -  -  -  -  -  -
0063 MYJOB        0.001004     0.112320     0.113325
0078 SOMEBAT1     0.000994     0.118045     0.119040
0066 GOODGUY      0.003275     0.128833     0.132108
0060 CICSRGNA     0.005703     0.202266     0.207969
0005 DUMPSRV      2.691197     1.319932     4.011129
                ------------ ------------ ------------
                  2.742155     2.564749     5.306904 (No. of CPUs in Systrace:  2)
```


In this data from SYSTRACE PERFDATA, one particular address space jumps out in terms of its CPU usage. You can see that the DUMPSRV address space is using large amounts of CPU time both in SRB mode and in task mode. Further down in this report there is a breakdown of this SRB and TCB usage. We will look at this in a moment.

Example: Interpreting PERFDATA

- **Previous slide showed DUMPSRV (SVC dump processing) using a large amount of CPU time relative to other address spaces**
  - **Could this be normal?** After all, we did just ask for a dump, so it stands to reason that we would see it doing a lot of work in the system trace table, right?
  - **No!** SVC dump processing knows that the debugger wants a picture of what happened BEFORE the dump, not during, so it captures the system trace table very early on in its processing
  - **So this high DUMPSRV activity in system trace warrants further investigation**

As debuggers, we must constantly assess whether data is meaningful or a red herring. It might be easy to dismiss high SVC dump CPU usage as a red herring, but the explanation on this slide shows that this finding is actually surprising in light of how SDUMP is designed to work. Therefore, it is worth investigating further.

Example: Drilling down in PERFDATA

- Let's look at the PERFDATA breakdown of DUMPSRV's SRB and TCB usage:

SRB breakdown by ASID:

```
ASID Jobname       SRB PSW      # of SRBs     Time
---- -------- ---------------- --------- ------------
0005 DUMPSRV  070C0000 85193190         2     0.000263
0005 DUMPSRV  070C4001 818EDC00         1     1.345179
0005 DUMPSRV  070C3000 814C641E         1     1.345754
                                          ------------
                                              2.691197
```

TCB breakdown by ASID:

```
ASID Jobname  TCB Adr  # of DSPs     Time
---- -------- -------- --------- ------------
0005 DUMPSRV  009FCC98         2     1.319932
```

- It's interesting and curious that nearly all of the DUMPSRV CPU time has accumulated across just 2 dispatches of a TCB plus 2 SRB dispatches

Normally time accumulates in small increments across many dispatches of a unit of work. The data being displayed in this section of the PERFDATA report is unusual because it shows a relatively large amount of CPU time being accumulated across just a couple dispatches of units of work. This suggests that the units of work, when they receive control, are tying up the processor for an extended period of time without giving up the CP.

Example: Using SYSTRACE to focus on DUMPSRV

- Let's focus on just the DUMPSRV address space, ASID 5
  - SYSTRACE ASID(5) TIME(LOCAL)

```
PR ASID WU-ADDR- IDENT  CD/D PSW----- ADDRESS-  UNIQUE-1 UNIQUE-2 UNIQUE-3  PSACLHS- PSALOCAL PASD SASD TIMESTAMP-LOCAL
                                               UNIQUE-4 UNIQUE-5 UNIQUE-6  PSACLHSE                    DATE-01/04/2012
00 0005 009FCC98  SVC     23 070C0000 851A5B7E  00F59B20 00000000 7FFC0E60  WTO/WTOR                     11:02:26.880656
00 0005 009FCC98  PC      ...   0     04580160              0030B          Storage  Obtain
00 0005 009FCC98  SPIN LKX/S         8190B276  01000002 00000081 021C3C04  88000001 00000000 0005 0005 11:02:27.929236
                                               00004000 000002F8 00FF15C2  00000000
01 0005 02491C80  SSRV   14B         04000001  00000000 00000000 00000000  IarV64   PageFix             11:02:28.199377
                                               00000000_7FFC6E08
                                               00000008_00300000
                                               00000000_00000100
00 0005 009FCC98  SPIN LKX/P         8190B276  0141F3DF 00000081 021C3C04  88000001 00000000 0005 0005 11:02:28.199378
                                               00004000 000002F8 00FF15C2  00000000
01 0005 02491C80  CALL       070C4001 81930D82  00011202 01000000          00000001 00FB5500 0004 0004 11:02:28.199379
                                                                           00000000
01 0005 02491C80  CLKC       070C4001 81930D82  00001004 009F0388     0039  00000001 00FB5500 0004 0004 11:02:28.199382
```

Spinning for the RSM address space lock …

… held by unit of work on CP 1

Note that the TCB address in the WU-ADDR column of the two highlighted SPIN entries is the same as we saw on the previous slide in the "TCB Breakdown by ASID" for DUMPSRV. Note also that the WU-ADDR in the trace entries for the work running on CP 1 is *not* a TCB address. (We know this because the address is an above the line address, and TCBs must live below the line.) This means that the work running on CP 1 is an SRB. This is consistent with the expectation set when we looked at the PERFDATA report which showed that both TCB-mode and SRB-mode activity was contributing to the large amount of CPU usage. If we were to look backwards in this system trace table, we would find an SRB or SSRB dispatch entry for the unit of work currently active on CP 1.

Example: Interpretation of SYSTRACE

- Work on CP 0 has made a STORAGE OBTAIN request to VSM
- VSM must have required RSM services because the SPIN entry shows the work on CP 0 spinning for an RSM lock
- Work on CP 1 completes an IARV64 PageFix request
- Then CP 0 must have been given the lock because the SPIN ends
- Other SPIN entries for the same lock existed in the system trace
- Conclusion: There is recurring contention for an RSM lock

```
PR ASID WU-ADDR- IDENT  CD/D PSW----- ADDRESS-  UNIQUE-1 UNIQUE-2 UNIQUE-3  PSACLHS- PSALOCAL PASD SASD TIMESTAMP-LOCAL
                                               UNIQUE-4 UNIQUE-5 UNIQUE-6  PSACLHSE                    DATE-01/04/2012
00 0005 009FCC98  SVC     23 070C0000 851A5B7E  00F59B20 00000000 7FFC0E60  WTO/WTOR                     11:02:26.880656
00 0005 009FCC98  PC      ...   0     04580160             0030B           Storage  Obtain
00 0005 009FCC98  SPIN LKX/S          8190B276  01000002 00000081 021C3C04  88000001 00000000 0005 0005 11:02:27.929236
                                               00004000 000002F8 00FF15C2  00000000
01 0005 02491C80  SSRV   14B          04000001  00000000 00000000 00000000  IarV64   PageFix             11:02:28.199377
                                               00000000_7FFC6E08
                                               00000008_00300000
                                               00000000_00000100
00 0005 009FCC98  SPIN LKX/P          8190B276  0141F3DF 00000081 021C3C04  88000001 00000000 0005 0005 11:02:28.199378
                                               00004000 000002F8 00FF15C2  00000000
01 0005 02491C80  CALL       070C4001 81930D82  00011202 01000000           00000001 00FB5500 0004 0004 11:02:28.199379
                                                                            00000000
01 0005 02491C80  CLKC       070C4001 81930D82  00001004 009F0388     0039  00000001 00FB5500 0004 0004 11:02:28.199382
```


The system trace table is a valuable tool for being able to piece together the order of events, and the details surrounding these events. From this small window in the system trace table, we are able to tell a story. The TCB at 9FCC98 on CP 0 is doing a STORAGE OBTAIN request at the same time that an SRB on CP 1 is processing an RSM PageFix request. VSM must have required RSM services in order to complete the STORAGE OBTAIN request, and these RSM services must have required the RSM Address Space lock. We are able to draw these conclusions from the fact that we see the TCB that did the STORAGE OBTAIN spinning for an RSM Address Space lock. We can also see that this spin lasted about 1.25 seconds. At that time we see that RSM has completed its PageFix request and released the lock that the TCB was spinning for. At this point the TCB is given the spin lock, and it is able to continue with the RSM processing that it requires. The bottom line here is that RSM lock contention is slowing down the progress of this TCB. Based on the PERFDATA about the DUMPSRV SRBs, we can theorize that the SRB is also suffering elongated run times due to RSM lock contention. This could be verified by checking for other SPIN entries in the system trace table.

Example: Conclusion (and success!)

- RETAIN data base search yielded an RSM APAR:

```
APAR= OA32947  SER=                    PR PERFM
IARV64 PAGEFIX TAKES EXCESSIVE TIME ON SYSTEMS WITH JUST OVER 2G
OF REAL
```

- An RSMDATA command verified that this was a small system (you may recall seeing in SYSTRACE PERFDATA that it had only 2 CPs) with 2.25Gig of real storage defined.

It's always a good thing when the clues we turn up as we review data gives us enough information to perform a meaningful search of the Retain data base. It's even better when we turn up a hit! APAR OA32947 was a good match for this problem.

BEAR - Breaking Event Address Register

- **A 64-bit register containing the address of the last instruction that causes a break in sequential execution**
  - For example, a branch or a LPSW instruction
- **Content of BEAR is stored in PSA when a program check occurred. This is propagated by z/OS to:**
  - SDWA (available in IPCS ST FAILDATA or VERBX LOGDATA)
  - RTM2WA (available in IPCS SUMM FORMAT)

BEAR is an enhancement in z/Architecture since the z9 machines (a while ago). Basically the machine remembers the address of the last instruction that causes a break in sequential execution (or in common terms, a branch) and surfaces this information in a program interrupt. If this program interrupts is not resolvable, resulting in an error condition, z/OS will save the contents of BEAR in the SDWA or the RTM2WA.

Debugging with BEAR

- **Diagnosing a wild branch**
  - Wild branch can result in ABEND0C1, ABEND0C4 or ABEND0C6
  - In most cases PSW at time of error does not point to anything meaningful
  - Together with the registers, BEAR can now be used to debug the problem
- **A clue for the module flow prior to an error**
  - BEAR is stored by H/W on any program check
  - So for any problem resulting from a program check, you can always use BEAR as a clue for the module flow prior to the error
- **But be careful**: BEAR may point to a system module such as a First Level Interrupt Handler or Dispatcher. They are not related.

BEAR is very useful in diagnosing a wild branch. In those situations, the PSW and registers at time of error may not clearing identify the culprit of the wild branch, but BEAR will. For other kinds of error that do not result from a bad branch, BEAR can also be used as a clue for the module flow prior to the error.

Finding BEAR in a dump

From IPCS ST FAILDATA or VERBX LOGDATA

```
TIME OF ERROR INFORMATION

  PSW: 07040001 80000000 00000000 0AF8D286
  INSTRUCTION LENGTH: 06   INTERRUPT CODE: 0010
  FAILING INSTRUCTION TEXT: 17884280 3015E320 60180004
  TRANSLATION EXCEPTION ADDRESS: 00000008_004FF800

  BREAKING EVENT ADDRESS: 00000000_0AF8C754
```

From IPCS SUMM FORMAT ASID(n)

```
RTM2WA: 7FFAFE10
   +0000  ID..... RTM2      ADDR..... 7FFAFE10  SPID..... FF        LGTH..... 0011F0
   +0014  VRBC..... 009FD550  ASC..... 00F882A0  CCF..... 84        CC..... 0C4000
.....
....   Lines omitted here
....
   +06C8  TRNE..... 00000000  072FF800
   +06D0  BEA..... 00000000  0AF8B552
   +06D8  PSW1..... 07040001  80000000  00000000  0AF8D180
```


You can find BEAR in the output of IPCS ST FAILDATA or VERBX LOGDATA, and also in the RTM2WA from IPCS SUMMARY FORMAT.

Debugging an ABEND0C1 with BEAR

From IPCS ST FAILDATA

```
TIME OF ERROR INFORMATION
PSW: 07850000 80000000 00000000 00000002
Instruction length: 02   Interrupt code: 0001
Failing instruction text: 00000000 000A0000 000130E1

Breaking event address: 00000000_00007F20
Registers 0-7
GR: 00000000 00007EF8 00000040 007D5D84  007D5D60 007FF448 007C7FE0 FD000000
AR: 00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
Registers 8-15
GR: 00007F22 007FF708 00000000 007FF448  965AB8B2 00006F60 80007F22 00000000
AR: 00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
```

From browsing storage

```
00007F1E                                          1BFF   |               .. |
00007F20   05EF5900   F0D24780   F01290EC   D00C1831   | ....0K..0...}... |
```

1BFF SR 15,15
05EF BALR 14,15

The above example is an ABEND0C1 due to a branch to location zero. R14 and R15 shows that there may be a branch and link instruction prior to 7F22 that is causing the problem. BEAR confirms that it is indeed a BALR instruction at 7F20.

Using BEAR in SLIP

- **BEAR is a symbolic in SLIP that represents the contents of BEAR at the time of the event causing the SLIP to match**
  - Useful in additional filtering for a SLIP
  - Meaningful only if the event is related to a program interrupt
  - PER SLIPs (IF, SA, SBT) events are actually program interrupts too
- **Examples:**
  - SLIP SET,C=0C1,DATA=(BEAR,EQ,00007F20),JOBNAME=XYZ,A=SVCD,END
    - Take a dump when a potential bad branch from 7F20 occurs
  - SLIP SET,IF,LMOD=(IGC0003E,0),DATA=(BEAR,EQ,00FFF324),A=SVCD,END
    - Take a dump when IGC0003E is called by the module at FFF324

BEAR is also a symbolic in SLIP. A symbolic is not a parameter, but a symbol that can be used when coding a SLIP trap. The BEAR symbolic denotes the contents of BEAR at the time of the event causing the SLIP to match. This can help in additional filtering of a SLIP trap. Note that PER events (instruction fetch, storage alterations or successful branches) are actually presented by hardware as program interrupts, so BEAR will be stored as well.

Reference Information

- **Manuals**
  - z/OS MVS IPCS Commands
  - z/OS MVS IPCS Customization
  - z/OS MVS IPCS User's Guide
  - z/OS MVS Diagnosis: Reference
  - z/OS MVS Diagnosis: Tools and Service Aids
  - z/OS MVS System Codes