Session 17907
z/OS Debugging:
Diagnosing Loops & Hangs

z/OS Core Technologies – August 13th, 2015

Patty Little  plittle@us.ibm.com
John Shebey  jshebey@us.ibm.com
IBM Poughkeepsie

SHARE Orlando, August 2015
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
Table of Contents

- Introduction ........................................ 4
- Diagnosing Loops ..................................... 10
- Loops and system trace .............................. 12
- Finding status for looping work .................. 19
- Diagnosing Hangs ..................................... 31
- IP ANALYZE RESOURCE ............................. 34
- Address space dispatchability ..................... 38
- Task dispatchability .................................. 45
- Hangs and SRBs ...................................... 53
Introduction
Section I - Diagnosing Loops and Hangs

There are many ways that an application or system can appear hung. Similarly, there are many factors which can cause or contribute to a hang. This presentation provides an overview of some types of hangs (including loops) and some steps for identifying what is causing the problem.

A looping application may be perceived as hung since a looping application will not be performing any significant work.

A function can be non-dispatchable at the address space level, or at the TCB level. Non-dispatchability bits being on prevent the address space or task from being dispatched. Alternatively, an RB may be suspended. Local lock contention is another common reason for a function to be not running.

When considering a hung application, it is necessary to be familiar with the application structure. Which TCBs drive which subfunctions? Are any key subfunctions waiting or suspended? If so, what event is the TCB waiting/suspended for? What process is responsible for making this event happen?

Sometimes a hung TCB is fully dispatchable, but it is not getting dispatched due to lack of available CPU. This could be the result of a tuning problem, or this could be due to a higher priority address space being stuck in a loop, thereby consuming extra CPU and starving lower priority address spaces.

---

**What is a "Hang"?**

**Definition:** No externally visible work being done by a process or function

**Possible triggers**

- Process is non-dispatchable
  - May have no work to do
  - May require a resource that is not available
  - May be waiting for an event that is not occurring
- Process is dispatchable but not getting CPU
  - May be a tuning problem
  - May be that a higher priority address space is looping or consuming excessive CPU
- Process is looping
In addition to getting a dump, never overlook the importance of LOGREC and SYSLOG! Often a loop or a hang is preceded by an abend that acted as a trigger or catalyst. Always check SYSLOG for relevant messages (e.g. IEA995I "symptom dump", D GRS output) and activity around the time of the onset of the loop or hang. Always check LOGREC for errors in relevant address spaces around the time of the onset of the loop or hang.

While a standalone dump is the documentation of choice for a system hang, sometimes system hangs can be diagnosed with an SVC dump.
Section I - Diagnosing Loops and Hangs

**Documentation for diagnosis of loops & hangs**

★ ★ Or better yet, take advantage of RTD! ★ ★

- Detects loop conditions such as HIGH CPU and TCB mode loops
- Detects hang conditions such as GRS and UNIX latch contention, ENQ contention, and local lock suspension

- `START HZR, SUB=MSTR`
- `F HZR, ANALYZE, OPTIONS=(DEBUG=(LOOP))`

- Produces a report of its findings (example on next slide)
- In the above example, if RTD detects a loop, the DEBUG option will cause it to automatically take a dump of the problem address space.

RTD = Run Time Diagnostics. Run RTD whenever your system is experiencing “sick but not dead” symptoms to do a one-minute (or less) diagnostic assessment. It checks for high CPU, loops, critical messages, GRS and UNIX latch contention, ENQ contention, and local lock suspension.
Section I - Diagnosing Loops and Hangs

RTD report example

EVENT 01: HIGH - HIGHCPU - SYSTEM: SY1 2010/12/21 - 13:51:33
ASID CPU RATE:99% ASID:002E JOBNAME:IBMUSERX
STEPNAME:STEP1 PROCSTEP: JOBID:JOB00045 USERID:IBMUSER
JOBSTART:2010/12/21 11:22:51
ERROR: ADDRESS SPACE USING EXCESSIVE CPU TIME. IT MIGHT BE LOOPING.
ACTION: USE YOUR SOFTWARE MONITORS TO INVESTIGATE THE ASID.

EVENT 02: HIGH - LOOP - SYSTEM: SY1 2010/12/21 - 13:51:14
ASID:002E JOBNAME:IBMUSERX TCB:004FF1C0
STEPNAME:STEP1 PROCSTEP: JOBID:JOB00045 USERID:IBMUSER
JOBSTART:2010/12/21 - 11:22:51
ERROR: ADDRESS SPACE MIGHT BE IN A LOOP.
ACTION: USE YOUR SOFTWARE MONITORS TO INVESTIGATE THE ASID.

When both a HIGHCPU and a LOOP condition are detected by RTD, the Job is very likely looping.
The system trace table is the best option for distinguishing a loop from a hang. A looping address space will have many entries, often composed primarily of EXT TIMR, EXT CLKC, and I/O interrupts, as well as DSP entries. A totally hung address space will have no entries in the system trace table.

Note that a hung address space may still have some work running in it. In such a case, the activity may be limited to timers being set (SVC 2F) and popping. Alternatively, there may be work running in the address space that is unrelated to the subfunction that is non-responsive. When debugging hangs, it is helpful to have a familiarity with the internal workings of the hung address space.
Diagnosing Loops
Steps for diagnosing a loop

Goal is to locate a PSW and register set that can be used to “pump code” to explain the loop

- Identify loop pattern in system trace table
  - Note ASID and Work Unit
  - Note PSW addresses
  - Note environmental information
    - PSW ASC mode (P, S, H, or AR mode)
    - Cross memory environment (PASID, SASID)
    - Local lock status

- Use trace info to locate GPRs, ARs, and matching PSW for the looping unit of work
  - Use PSW address to identify looping code
  - Use regs to pump the code, determine reason for loop
Recognizing enabled loops in SYSTRACE

- Some loops are easy to pick out in the trace table by their repetitive pattern of events:

<table>
<thead>
<tr>
<th>0001 001A 008F8238</th>
<th>PC ... 0</th>
<th>25900040</th>
<th>0030B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001 001A 008F8238</td>
<td>SSRV 132</td>
<td>00000000</td>
<td>00000672</td>
</tr>
<tr>
<td>0001 001A 008F8238</td>
<td>PR ... 0</td>
<td>25900040</td>
<td>015EC052</td>
</tr>
<tr>
<td>0001 001A 008F8238</td>
<td>PC ... 0</td>
<td>2590006E</td>
<td>00311</td>
</tr>
<tr>
<td>0001 001A 008F8238</td>
<td>SSRV 133</td>
<td>00000000</td>
<td>00000603</td>
</tr>
<tr>
<td>0001 001A 008F8238</td>
<td>PR ... 0</td>
<td>2590006E</td>
<td>015EC052</td>
</tr>
<tr>
<td>0001 001A 008F8238</td>
<td>PC ... 0</td>
<td>25900040</td>
<td>0030B</td>
</tr>
<tr>
<td>0001 001A 008F8238</td>
<td>SSRV 132</td>
<td>00000000</td>
<td>00000672</td>
</tr>
<tr>
<td>0001 001A 008F8238</td>
<td>PR ... 0</td>
<td>25900040</td>
<td>015EC052</td>
</tr>
<tr>
<td>0001 001A 008F8238</td>
<td>PC ... 0</td>
<td>2590006E</td>
<td>00311</td>
</tr>
<tr>
<td>0001 001A 008F8238</td>
<td>SSRV 133</td>
<td>00000000</td>
<td>00000603</td>
</tr>
<tr>
<td>0001 001A 008F8238</td>
<td>PR ... 0</td>
<td>2590006E</td>
<td>015EC052</td>
</tr>
</tbody>
</table>

- Other loops are a little trickier to recognize
In this presentation we will not discuss how to determine whether an SRB is preemptable or non-preemptable. For our purposes it is sufficient to know that if an SRB/SSRB gets interrupted and loses the processor, then it must have been a preemptable-type SRB/SSRB.
A TCB is always preemptable. This means that, when it gets interrupted, it may lose the CP and have to be redispatched before it can run again. Often when a TCB is interrupted, it does NOT lose the CP but rather is allowed to continue running on the same CP once the interrupt has been handled.

When you see a DSP entry, this means the TCB was found on the WUQ (Work Unit dispatching Queue) and has gotten redispatched. It can get redispatched on a different CP than the one on which it was running prior to the interrupt.

When a TCB is running, the Work Unit address that is traced is the TCB address. We will see that for SRBs, the traced Work Unit address is actually the address of the WEB representing that SRB/SSRB.
A preemptable SRB, as its name implies, may have to give up the processor on an interrupt, and wait in line on the WUQ dispatching queue for another opportunity to be dispatched. Therefore, like TCBs, when a preemptable SRB is looping, the loop will travel across processors.

When a preemptable SRB gets interrupted, its status (PSW, registers, cross memory environment) needs to be saved. The operating system obtains an SSRB control block to hold this information. (This is the same type of control block as is used for an SRB, preemptable or non-, when it gets suspended.) This is why we are seeing SSRB entries rather than SRB entries in the system trace output.

WEB = Work Element Block. WEB control blocks represent units of work. They will point to a TCB, an SRB, or an SSRB. WEBs representing ready units of work get queued to a WUQ dispatching queue in priority order. When an SRB (preemptable or non-) is running, the Work Unit address that is traced is the WEB address for that SRB/SSRB.

<table>
<thead>
<tr>
<th>PR</th>
<th>ASID</th>
<th>WU-Addr-</th>
<th>Ident</th>
<th>CD/D</th>
<th>PSW-</th>
<th>Address-</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>0025</td>
<td>0264BB00</td>
<td>SSRB</td>
<td>00000000_076B5624</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0001</td>
<td>0025</td>
<td>0264BB00</td>
<td>EXT</td>
<td>CLKC</td>
<td>00000000_076B5F20</td>
<td></td>
</tr>
<tr>
<td>0001</td>
<td>0025</td>
<td>0264BB00</td>
<td>EXT</td>
<td>TIMR</td>
<td>00000000_076B5686</td>
<td></td>
</tr>
<tr>
<td>0003</td>
<td>0025</td>
<td>0264BB00</td>
<td>SSRB</td>
<td>00000000_076B5686</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0003</td>
<td>0025</td>
<td>0264BB00</td>
<td>I/O</td>
<td>0265e</td>
<td>00000000_076B5634</td>
<td></td>
</tr>
<tr>
<td>0003</td>
<td>0025</td>
<td>0264BB00</td>
<td>EXT</td>
<td>TIMR</td>
<td>00000000_076B5798</td>
<td></td>
</tr>
<tr>
<td>0001</td>
<td>0025</td>
<td>0264BB00</td>
<td>SSRB</td>
<td>00000000_076B5798</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

As a result of the system trace output, we see EXT and I/O interrupt entries, but instead of DSP dispatch entries, we have SSRB dispatch entries.
Section I - Diagnosing Loops and Hangs

Non-preemptable SRB mode loop

IP SYSTRACE ASID(X’26’) TI(LO)

<table>
<thead>
<tr>
<th>PR</th>
<th>ASID WU-Addr-</th>
<th>Ident</th>
<th>CD/D</th>
<th>PSW-----</th>
<th>Address-</th>
<th>PR</th>
<th>ASID WU-Addr-</th>
<th>Ident</th>
<th>CD/D</th>
<th>PSW-----</th>
<th>Address-</th>
</tr>
</thead>
<tbody>
<tr>
<td>0002-0026 07F0BF00</td>
<td>EXT</td>
<td>CLKC</td>
<td>00000000_0767B89C</td>
<td>07041000 80000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0002-0026 07F0BF00</td>
<td>SSRV</td>
<td>110</td>
<td>810AEE00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0002-0026 07F0BF00</td>
<td>EXT</td>
<td>TIMR</td>
<td>00000000_0767CA26</td>
<td>07040000 80000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0002-0026 07F0BF00</td>
<td>EXT</td>
<td>CLKC</td>
<td>00000000_0767B84E</td>
<td>07041000 80000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0002-0026 07F0BF00</td>
<td>SSRV</td>
<td>120</td>
<td>81360308</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0002-0026 07F0BF00</td>
<td>EXT</td>
<td>CLKC</td>
<td>00000000_0767BFD2</td>
<td>07042000 80000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0002-0026 07F0BF00</td>
<td>I/O</td>
<td>0265E</td>
<td>00000000_0767C002</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Sometimes there is some “clutter” under the EXT trace entries. This is coming from code running under timer DIEs (disabled interrupt exits).

SRB stays on same CP.

Work Unit address is that of a WEB.

Note there are no dispatch trace entries of any kind.

Non-preemptable SRBs cannot lose the processor. You will not see DSP or SSRB trace entries in the pattern of the loop. The SRB will never get preempted and have to be redispached; therefore, the loop will stay on the same CP rather move around as was the case with the TCB and preemtptable SRB mode loops.
Simplifying the trace output

- With so many CPs (and therefore so much parallel activity) on some machines, it can be difficult to pick out a loop

- SYSTRACE offers several filtering options
  - TCB: SYSTRACE ASID(‘yy’) TCB(‘zzzzzz’)
  - SRB: SYSTRACE WEB(‘zzzzzzzzz’)
  - Non-preemptable SRB
    SYSTRACE ASID(‘yy’) CPU(‘zzzz’)

Course materials may not be reproduced in whole or in part without the prior written permission of IBM. (c) Copyright IBM 2009.
Note **ASID** and **WU (Work Unit) address** (TCB/WEB)
- If non-preemptable SRB, note **Processor** number (CP)
- If TCB, note **last bit** of **PSACLHS** (upper word)
  - If on, then work unit holds a local lock
    - **PSALOCAL** indicates ASCB whose local lock is held by this work unit
    - **PSALOCAL=0** indicates holding home address space’s local lock

The last bit of PSACLHS can also be used to determine whether an SRB is holding a local lock. However, this information is not needed to locate the status information of an SRB.
Finding Status: TCB w/o lock

- SUMM FORMAT ASID(X‘yy’)  [ASID from systrace]
- FIND ‘TCB: 00zzzzzz’  [TCB from systrace]
- General Purpose Registers
  - TCB, under heading “64-Bit GPRs from TCB/STCB”
- Access Registers
  - TCB’s STCB+X’30’
- PSW
  - Current RB’s XSBOPS16 at XSB+X‘F0’
    (Current RB is last one formatted under TCB)
- PASID and SASID
  - Current RB’s XSBPASID and XSBASID respectively
  - SASID: XSB+X’D6’  PASID: XSB+X’CE’

Status: PSW, General Purpose Registers, Access Registers, Cross Memory environment. PASID and SASID stand for Primary ASID and Secondary ASID respectively.
# Registers in TCB/STCB

TCB: 005F81A0

- 64-Bit GPRs from TCB/STCB
- Left halves of all registers contain zeros

<table>
<thead>
<tr>
<th>0-3</th>
<th>07812870</th>
<th>08DCE428</th>
<th>00000032</th>
<th>07812870</th>
</tr>
</thead>
<tbody>
<tr>
<td>4-7</td>
<td>00000003</td>
<td>08DCFC67C0</td>
<td>07639CCE</td>
<td>084DCFC0A1</td>
</tr>
<tr>
<td>8-11</td>
<td>008DCFC61D</td>
<td>008DCE8A8</td>
<td>0076C806C</td>
<td>00000000</td>
</tr>
<tr>
<td>12-15</td>
<td>FFFFFFFF</td>
<td>08DCE8A8</td>
<td>876C7650</td>
<td>00000000</td>
</tr>
</tbody>
</table>

STCB: 7FF80420

| +0030 | AR0...... | 005FDD40 | AR1...... | 00000000 | AR2...... | 00000000 |
| +003C | AR3...... | 00000000 | AR4...... | 00000000 | AR5...... | 00000000 |
| +0048 | AR6...... | 00000000 | AR7...... | 00000000 | AR8...... | 00000000 |
| +0054 | AR9...... | 00000000 | AR10...... | 00000000 | AR11...... | 00000000 |
| +0060 | AR12...... | 00000000 | AR13...... | 00000000 | AR14...... | 00000000 |
| +006C | AR15...... | 00000000 | LSSD...... | 7FF82CA0 | LSDP...... | 7F54A138 |

General purpose registers are formatted under the TCB, access registers under the STCB.
PSW and XMEM info in XSB

TCB: 005F81A0
   +0000   RBP...... 005F040   PIE...... 00000000   DEB...... 00000000
   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 217 LINE(S) NOT DISPLAYED

PRB: 005F040
   -0020   XSB...... 7FFFD10   FLAGS2... 80   RTPSW1... 00000000
   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 29 LINE(S) NOT DISPLAYED

XSB: 7FFFD10
   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 22 LINE(S) NOT DISPLAYED
   +00CC   KM....... 00C0   SASID..... 0027   PINS..... 00000006
   +00D4   AX....... 0005   PASTID..... 0027
   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 2 LINE(S) NOT DISPLAYED
   +00F0   OPS16.... 07041000    80000000    00000000   0767E84E

- TCB/RBP points to the “top” or “current” RB, which is the last RB formatted under this TCB in SUMM FORMAT.
- The corresponding XSB contains PSW and XMEM information.
Finding Status: TCB with lock

- If PSALOCAL non-zero:
  - CBF xxxxxxx STR(ASCB)
  - FIND ASID
- SUMM FORMAT ASID(X’yy’)
  - If PSALOCAL=0: X’yy’ = Home ASID [ASID from systrace]
  - If PSALOCAL non-zero: X’yy’ is ASID found above
- FIND IHSA  [field in ASXB]
- CBF X’zzzzzz’ STR(IHSA) ASID(X’yy’)
  - PSW, GPRs, Ars
  - Note XSB address with IHSA
- CBF X’aaaaaaaa’ STR(XSB) ASID(X’yy’)
  - PASID and SASID

Status: PSW, General Purpose Registers, Access Registers, Cross Memory environment
# Section I - Diagnosing Loops and Hangs

## PSW, Regs in IHSA

<table>
<thead>
<tr>
<th>Offset</th>
<th>Register</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>+0000</td>
<td>AXEB.....</td>
<td>00000000</td>
</tr>
<tr>
<td>+0000</td>
<td>ASXB.....</td>
<td>00000000</td>
</tr>
<tr>
<td>+0000</td>
<td>FTCB.....</td>
<td>00000000</td>
</tr>
<tr>
<td>+0000</td>
<td>00FDDA0</td>
<td>00FDDA0</td>
</tr>
<tr>
<td>+000C</td>
<td>TCBS.....</td>
<td>00040000</td>
</tr>
<tr>
<td>+000C</td>
<td>00F81A0</td>
<td>00F81A0</td>
</tr>
<tr>
<td>+0010</td>
<td>MPST.....</td>
<td>00000000</td>
</tr>
<tr>
<td>+0010</td>
<td>00000000</td>
<td>00000000</td>
</tr>
<tr>
<td>+001C</td>
<td>SAF.....</td>
<td>00000000</td>
</tr>
<tr>
<td>+001C</td>
<td>IHSA.....</td>
<td>005FE470</td>
</tr>
<tr>
<td>+001C</td>
<td>FLSA.....</td>
<td>00000000</td>
</tr>
</tbody>
</table>

### IHSA: 005FE470

<table>
<thead>
<tr>
<th>Offset</th>
<th>Register</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>+0000</td>
<td>CFPUT.....</td>
<td>00000000</td>
</tr>
<tr>
<td>+0000</td>
<td>00770000</td>
<td>00770000</td>
</tr>
<tr>
<td>+000C</td>
<td>NTCH.....</td>
<td>00F81A0</td>
</tr>
<tr>
<td>+000C</td>
<td>00F81A0</td>
<td>00F81A0</td>
</tr>
</tbody>
</table>

#### General purpose register values

<table>
<thead>
<tr>
<th>Offset</th>
<th>Register</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-3</td>
<td>D3D3D7E2</td>
<td>7FF59F50</td>
</tr>
<tr>
<td>4-7</td>
<td>7FF7D46C</td>
<td>01BC20F8</td>
</tr>
<tr>
<td>8-11</td>
<td>8123D180</td>
<td>7FF7D400</td>
</tr>
<tr>
<td>12-15</td>
<td>01529240</td>
<td>81529200</td>
</tr>
<tr>
<td></td>
<td>XSB......</td>
<td>7FFFD430</td>
</tr>
<tr>
<td></td>
<td>FLGS.....</td>
<td>00000000</td>
</tr>
</tbody>
</table>

### Access register values

<table>
<thead>
<tr>
<th>Offset</th>
<th>Register</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-3</td>
<td>00000000</td>
<td>00000000</td>
</tr>
<tr>
<td>4-7</td>
<td>FFFFFFFF</td>
<td>FFFFFFFF</td>
</tr>
<tr>
<td>8-11</td>
<td>FFFFFFFF</td>
<td>FFFFFFFF</td>
</tr>
<tr>
<td>12-15</td>
<td>FFFFFFFF</td>
<td>00000000</td>
</tr>
<tr>
<td></td>
<td>FFFFFFFF</td>
<td>00000000</td>
</tr>
</tbody>
</table>

© 2016 IBM Corporation

SHARE Orlando, August 2015

Course materials may not be reproduced in whole or in part without the prior written permission of IBM. (c) Copyright IBM 2009.
This XSB lives in the same address space as the IHSA.
Section I - Diagnosing Loops and Hangs

Finding Status: Preemptable SRB

- CBF X’xxxxxxxx’ STR(WEB)  [WEB address from systrace WU-addr]
- FIND UPTR

WEB: 0257B200
+0000 WEB....... WEB  FLAG1.... 0000  FLAG2.... 08
+0007 TYPE..... 14  LOCK..... 00000000  WUQF..... 0772A600
+0010 CMajor_B. 00FE  CMINOR_B. 0000  HASCB.... 00F9A380
+0018 UPTR..... 0269F020  UNEXT.... 0772A600  UPREV.... 00000000

- CBF X’yyyyyyyy’ STR(SRB)  [where yyyyyyyyy is UPTR value]
  - Note: STR(SRB) can be used for both SRB’s and SSRB’s
  - Locate PSW, GPRs, and ARs
  - FIND ‘XSB’  [to get address of XSB]
- CBF X’zzzzzzzz’ STR(XSB)
  - Get PASID and SASID

Status: PSW, General Purpose Registers, Access Registers, Cross Memory environment
SSRBCPSW is the “scrunched” version of SSRBPSW16. Technically SSRBPSW16 is the correct PSW to use now that there is limited execution allowed above the bar. However, the number of exploiters is in fact so small that debuggers can get away with still using SSRBCPSW virtually 100% of the time. Several IPCS reporting execs still format SSRBCPSW instead of SSRBPSW16.
PSW, Regs, XMEM info in XSB

<table>
<thead>
<tr>
<th>XSB: 0269F0D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>+00C0 TRNE..... 00000000 00000000   SINS..... 00000013</td>
</tr>
<tr>
<td>+00CC KM........ 8000 SASID.... 0026  PINS..... 00000013</td>
</tr>
<tr>
<td>+00D4 AX........ 0005 PASID.... 0026</td>
</tr>
</tbody>
</table>
Finding Status: Non-preemptable SRB

- CBF PSAx  [where x is Processor from systrace]
- FIND SCFS
- CBF X’yyyyyyyyy’ STR(SCFS)  [where yyyyyyyyy is SCFS addr]
  - GPRs in SCFSX1G0
  - Access Registers in SCFSX1A0
  - PSW in SCFSX1P
  - Cross memory environment in SCFSX1SS (SASID) and SCFSX1PS (PASID)

Status: PSW, General Purpose Registers, Access Registers, Cross Memory environment
PSW, Regs, XMEM info in SCFS

SCFS: 020FC100
- - - - - - - - - - - - - - - - - - 14 Line(s) not Display
+00D0 X1G0..... 0819B088 X1G1..... 08D66460 X1G2..... 00000024
+00DC X1G3..... 07627D60 X1G4..... 07627DF8 X1G5..... 0819B088
+00E8 X1G6..... 07630F84 X1G7..... 00000000 X1G8..... 07627DF8
+00F4 X1G9..... 08D666B0 X1GA..... 076C806C X1GB..... 00000000
+0100 X1GC..... 08D674C8 X1GD..... 08D666B0 X1GE..... 0771434E
+010C X1GF..... 076BE930
+0110 X1A0..... 00000000 X1A1..... 00000000 X1A2..... 00000000
+011C X1A3..... 00000000 X1A4..... 00000000 X1A5..... 00000000
+0128 X1A6..... 00000000 X1A7..... 00000000 X1A8..... 00000000
+0134 X1A9..... 00000000 X1A1A..... 00000000 X1A1B..... 00000000
+0140 X1A1C..... 00000000 X1A1D..... 00000000 X1A1E..... 00000000
+014C X1A1F..... 00000000 XRSA..... 00000000 00000000 00000000
- - - - - - - - - - - - - - - - - - - - 15 Line(s) not Display
+0250 P161..... 07046001 80000000 00000000 076C23E8
- - - - - - - - - - - - - - - - - - - - 30 Line(s) not Display
+03D0 X1GN..... 00000007 X1PK..... 0000 X1GS..... 0026
+03D8 X1PN..... 00000007 X1AX..... 0005 F1PS..... 0026
- - - - - - - - - - - - - - - - - - - - - - - 78 Line(s) not Display
Section I - Diagnosing Loops and Hangs

Pumping the code

- Do **IPCS WHERE** or **IPCS BROWSE** against the PSW address
  - If PSW address points to private storage, **make sure you specify the PASID as the ASID**
    - E.g. IP WHERE xxxxxxxx ASID(X'yy')
- Before using your registers, check **PSW ASC mode bits** (bits 16 and 17)
  - 00 – Data reference is to primary address space
  - 10 – Data reference is using access registers
  - 01 – Data reference is to secondary address space
  - 11 – Data reference is to home address space
Diagnosing Hangs
Steps for diagnosing a hang

Goal is to locate a unit of work that is the bottleneck, and to use its status (PSW, registers) and related information to explain why it is not progressing.

- Identify pivotal unit of work
- Gather dispatchability information about unit of work
  - If waiting, who did the WAIT?
  - If suspended, who did the SUSPEND?
  - If PAUSEd, who did the PAUSE?
  - If dispatchable, why isn’t it running?
System hangs

System hang
- Either the whole system, or else a major subset, is not functioning
- Ideal documentation is a SADump but some diagnoses can be made using a console dump

System hang diagnosis comes in 2 flavors: 😊 / 😞
- IP ANALYZE RESOURCE helps 😊
  - Report highlights contention and identifies the “bottlenecking” unit of work
- IP ANALYZE RESOURCE doesn’t help 😞
  - Need to work harder for answer
  - Consider what address spaces aren’t running
    - Verify their activity (or lack thereof) in system trace
      IP SYSTRACE JOBNAME(jjjjjjjj) TI(LO)
    - Explore their dispatchability
IP ANALYZE RESOURCE

- Use IP ANALYZE RESOURCE to identify contention
  - Identifies resource
  - Identifies owner and owner’s dispatchability status
  - Identifies contenders

- Report may call out multiple points of contention
  - Look for key system resources such as a local lock for a critical system address space
  - Look for long lists of contenders
  - Look for contention involving jobs you know to be hung

- **NOTE:** While ANALYZE RESOURCE is the “go-to” command for system hangs, it can be useful for address space hangs as well.
Details in ANALYZE RESOURCE report

- Examples of contention identified by ANALYZE RESOURCE
  - Suspend lock contention (LOCAL/CML/CMS)
    - Note: report calls this out even if no contenders
  - I/O device
  - ENQ resource (Major/Minor)
  - Page fault
  - Latches
    - Latch control blocks live in the latch set owner's address space.
    - Contention will only show in ANALYZE RESOURCE if owner's address space dumped.

- Examples of resource owner status identified by ANALYZE RESOURCE
  - Suspended or waiting
  - Interrupted but dispatchable
  - Executing on a CP
ASID 4 is the TRACE address space. A trace table is “snapped” as part of dump processing, and doing this requires the local lock of the TRACE address space.

**ANALYZE RESOURCE examples**

RESOURCE #0004:
NAME=LOCAL LOCK FOR ASID 00BA

RESOURCE #0004 IS HELD BY:

JOBNAME=ABC  ASID=00BA  SSRB=1A31940C
DATA=INTERRUPTED AND NOW DISPATCHABLE

RESOURCE #0002:
NAME=LOCAL LOCK FOR ASID 0004

RESOURCE #0002 IS HELD BY:

JOBNAME=*MASTER*  ASID=0001  SSR=00000000  CPU=76
DATA=CURRENTLY RUNNING ON CPU 26

ASID 4=TRACE. Typical and not a concern in SVC dumps.
ANALYZE RESOURCE examples

RESOURCE #0002:
NAME=MAJOR=CATLGRES MINOR=CASE SCOPE=SYSTEM

RESOURCE #0002 IS HELD BY:
JOBNAME=CATALOG ASID=0031 TCB=008AC680

RESOURCE #0002 IS REQUIRED BY:
JOBNAME=PBLPROG ASID=0117 TCB=008D1210

No dispatchability status; Perhaps CATALOG job was not in dump.
Checking address space dispatchability

- Check address space dispatchability if:
  - Problem is a hung address space
  - ANALYZE RESOURCE didn’t help identify the source of a system hang

- Steps for checking address space dispatchability
  - Check address space level non-dispatchability bits
  - Check task level non-dispatchability bits for key TCBs
    - Identifying key TCBs may require some inside knowledge of address space
    - For hangs during CANCEL or job shutdown, last TCB is often the bottleneck
  - Check RB level non-dispatchability indicators
  - Validate whether unit of work is on/off the WUQ dispatch queue

- What if address space is hung due to an SRB not running?
  - Locate SRB/SSRB on address space’s "in flight" queue
  - Validate whether unit of work is on/off the WUQ dispatch queue

Note: IP SUMM FORMAT ASID(X`yy`) to view ASCB/TCBs
ASCB non-dispatchability bits

Located in ASCBDSP1 at ASCB+X’72’, length 1 byte

Common settings:
- X’00’  Address space is dispatchable
- X’80’  Address space quiesced due to SVC dump in progress - not a problem!
- X’18’  Contact Supervisor L2 (compID 5752SC1C5)
- X’40’  Address space being terminated (MEMTERM)
- X’10’  Address space logically swapped out
         OR
         TCB within address space has issued
         STATUS STOP of SRBs
Address space nondispatchability bits in ASCBDSP1 are described verbally immediately underneath the formatted ASCB. Note that if no bits are on, no verbiage will appear.
Troubleshooting
ASCB non-dispatchability

- X’40’ Address space being terminated (MEMTERM)

- Memterm is usually a quick process ... SO ...
  - Address space in memterm processing => memterm hung

- To debug:
  - Get a console dump of ASID1
    (memterm is driven from ASID1)
    - IP SUMM FORMAT ASID(1)
  - FIND IEAVTMTR (to locate task driving memterm)
    - Eyecatcher appears under first RB belonging to IEAVTMTR TCB
  - Verify that Reg1 in first RB matches our ASCB address
    - If not, repeat FIND IEAVTMTR looking for other memterm TCBs
  - Apply TCB non-dispatchability checks against IEAVTMTR TCB
Troubleshooting ASCB non-dispatchability

- X‘10’ Address space swapped out OR “STATUS STOP-ed”

- IP VERBX SRMDATA
  - FIND ‘ASID xxxx’ to locate swap status of address space
  - If logically swapped, for what condition?
    - WAITing – Apply TCB non-dispatchability checks to key TCBs in address space
    - Unilaterally swapped – Possible MPL issue
    - If not logically swapped, then address space must be STATUS STOP-ed
Example: VERBX SRMDATA

<table>
<thead>
<tr>
<th>JOB</th>
<th>INIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASID</td>
<td>007B</td>
</tr>
<tr>
<td>OUCB</td>
<td>04FCB500</td>
</tr>
</tbody>
</table>

**LS WAIT QUEUE**

- +10 (LSW) LOGICALLY SWAPPED
- +11 (PVL) PRIVILEGED PROGRAM
- +29 (SRC) SWAP OUT REASON: LONG WAIT

(ASCBRSMME) RAX ADDRESS IS 05885BA8
SERVICE CLASS = SYSSTC
WORKLOAD = SYSTEM
INTERNAL CLASS= $SRMGOOD
PERIOD = 01
Troubleshooting ASCB non-dispatchability

- X'10'
  Address space “STATUS STOP-ed”

- A TCB in the address space has requested STATUS STOP of SRBs
  - z/OS stops all SRB and TCB work in this address space except for requesting TCB
  - To diagnose, we need to identify requesting TCB
    - All TCBs but one will have TCBSRBND bit on (TCB + X'AF', bit X'20' )
    - Apply TCB non-dispatchability checks to TCB with TCBSRBND off
Checking task dispatchability

- For key TCBs, or for each TCB in address space:
  - Is the TCB on the WUQ (dispatching queue)?
    IP IEAVWEBI WUQ
  - Yes, then why isn’t it running?
    - WUQ backed up?
      Further check IP IEAVWEBI WUQ
    - Dispatching priority issue?
      - How does our TCB’s priority compare to others on WUQ
        Further check IP IEAVWEBI WUQ
      - What ASIDs are running in system trace? (SYSTRACE ALL)
      - What is their dispatching priority compared to ours?
        (SUMM FORMAT ASID(X‘yy’) ; F DPH [within ASCB] )
  - Is something looping?
    Check for loops in system trace (SYSTRACE ALL)
  - No, then check TCB non-dispatchability indicators
### SUMMARY BY WUQ, SORTED BY TOTL:

<table>
<thead>
<tr>
<th>WUQ</th>
<th>TOTL</th>
<th>PROC</th>
<th>CPUMASK</th>
</tr>
</thead>
<tbody>
<tr>
<td>10309A800</td>
<td>156</td>
<td>CF</td>
<td>FE000000 00000000</td>
</tr>
<tr>
<td>10309A200</td>
<td>2</td>
<td>CF</td>
<td>00000000 00000000</td>
</tr>
<tr>
<td>10309A900</td>
<td>2</td>
<td>ZIIP</td>
<td>01840000 00000000</td>
</tr>
</tbody>
</table>

---

There are actually multiple WUQs. Section shows number of ready-to-run work units on each WUQ; in this example, 156 indicates a busy system with some backup.

---

### SUMMARY BY ASID, SORTED BY TOTL:

<table>
<thead>
<tr>
<th>ASID</th>
<th>JOBNAME</th>
<th>TOTL</th>
<th>ZAAP</th>
<th>ZIIP</th>
<th>TCB</th>
<th>SSRB</th>
<th>SRB</th>
<th>MSRB</th>
<th>ESRB</th>
<th>PSRB</th>
<th>PSRB</th>
<th>EXIT</th>
<th>CMLP</th>
</tr>
</thead>
<tbody>
<tr>
<td>0254</td>
<td>WXYZ</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1006F</td>
<td>ABCD</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>-------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>-----</td>
<td>------</td>
<td>-----</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>TOTL</td>
<td>160</td>
<td>0</td>
<td>2</td>
<td>40</td>
<td>0</td>
<td>95</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

### DETAILED INFORMATION FOR WUQ 0309A200, SORTED BY WEB DPH:

| WFR#: | TYPE: | WEB DPH | WUQ#: | CIAS| ASCB | | W#: | PROMOTE | | JORMAK | AST| DPH |
|-------|-------|---------|-------|-----|-----| |     |         | |       |   |     |
| 1045C3100 | SRB | 0FF8300 | 0309A200 | CF | 00F2300 | | 02157FA0 | | | | |
| 1390BE80 | TCB | 0FF240FF | 0309A200 | CF | 00F12480 | | 000A2CF0 | | | | |

---

TCB's address es omitted -- -- -- -- -- --
Before we start checking the TCB...

- As with a loop, we will be looking for status information (PSW, regs, xmem environment)
- Reminder:
  - **Status for an unlocked TCB** is saved as follows:
    - PSW in top RB’s XSBOPS16
    - GPRs in TCB; ARs in STCB
    - XMEM info (PASID, SASID) in XSB pointed to by TCB
  - **Status for a locally locked TCB** is saved in the IHSA of the address space whose lock is held
    - PSW, GPRs, and ARs in IHSA
    - XMEM info in XSB pointed to by IHSA
  - Instruction execution occurs in the primary address space
- If a TCB is locally locked
  - TCBLLH (+X’114’, X’01’ bit) on
  - TCBXLAS (+X’E8’) holds addr of ASCB whose lock is held
TCB non-dispatchability bits

Located in TCB at:
TCBFLGS (+1D) – last two bytes of 5-byte field
TCBNDSP (+AC) – 4 bytes

Common settings:
- FLGS = xxxxxx04 01  NDSP = 00002000  Top RB in a wait, check TCBNDSP
  Task non-dispatchable for SVCDump
- FLGS = xxxxxx00 01  NDSP = 00002000  Check TCBNDSP
  Task non-dispatchable for SVCDump
- FLGS = xxxxxx04 00  Top RB in a wait, TCBNDSP = 0 (SAdump)
- FLGS = xxxxxx00 00  No TCB non-dispatchability bits set

For other bit settings, see TCB mapping in MVS Data Areas

NOTE: TCB SUSPEND and PAUSE states are reflected elsewhere.
The last bit of TCBFLGS5 (which is the last byte in TCBFLGS) is a summary bit. If any bits are on in TCBNDSP, then this summary bit is turned on.

When SVC dump processing gathers local storage, it sets TCBs non-dispatchable in order to get a more stable picture. To make then non-dispatchable, dump processing calls a system service called STATUS who turns on the X'20' bit at TCB+x'AE' in each TCB. Therefore, when you see a X'00002000' in the TCBNDSP field at +AC, this is an effect of the dump in progress and is not relevant to debugging of the hang.
If TCB is waiting....

- Get PSW from XSBOPS16 of top RB’s XSB
- IP WHERE or IPCS Browse the PSW address, making sure you use the correct PASID
  - IEAVEWAT? Then WAIT was PC-entered
    - Find last LSE linkage stack entry (between TCB and STCB)
    - LSE TARG field should contain 0000030D
    - LSE PSWE will point to who issued the PC 30D WAIT
  - Otherwise, WAIT was SVC- or branch-entered
    - XSBOPS16 points to the issuer of the WAIT
TCB suspended?

Is TCB suspended?

- Get first byte of RBLINK from TCB’s top RB
  - If X’01’ then TCB is suspended
    - If TCB is unlocked, get suspend PSW from top RB’s XSBOPS16
    - If TCB is locked, get suspend PSW from IHSACPSW
    - IP WHERE or IPCS Browse the PSW address, making sure to use the correct PASID

NOTE: The SUSPEND PSW typically points right after a BALR instruction. If this is not the case, the SUSPEND could be due to a translation exception (e.g. page fault). Check for a non-zero XSBRTRNE (XSB+X’C0’) that matches the instruction base register or the instruction address.
TCB not dispatchable: other checks

- Get PSW from XSBOPS16 of top RB’s XSB or from IHSACPSW
- IP WHERE or IPCS Browse the PSW address, making sure to use the correct PASID
  - IEAVEPS1? => TCB is PAUSEd
    - Reg13 from TCB points to standard register save area
    - Reg13+C contains address where PAUSE was issued (use PASID when mapping return address to code)
  - IEAVESLK? => TCB is suspended for a lock
    - Reg14 from TCB/IHSA indicates caller
    - Did you check IP ANALYZE RESOURCE?
Address space hang due to SRB/SSRB

- Sometimes a "missing" SRB/SSRB causes an address space to hang

- Address space has a queue of in-flight SRBs
  - May be dispatchable (e.g. on WUQ) – **IP IEAVWEBI WUQ**
  - May be delayed/suspended for local lock – **ANALYZE RESOURCE**
  - May be suspended for page fault or other translation
  - May be WAITing
  - May be suspended explicitly by owner (SSRB)
  - May be PAUSEd

- Use **IP IEAVWEBI SRB ASID(xx)** to format an address space’s “in flight” SRBs and SSRBs
  - Use SRBEPA to recognize “missing” SRB
  - Need to do: **CBF ssrbaddr STR(SRB)** to get SSRB’s EPA
This is output from the IPCS command: IEAVWEBI SRB ASID(140) where 140 is a hexadecimal ASID number. This report shows in flight SRBs/SSRBs associated with ASID X'140'. When SRBs/SSRBs are not hung, their status is changeable. This means that in an SVC dump, you may see inconsistent data between the IEAVWEBI SRB ASID(xx) report which formats in flight SRBs/SSRBs, and the IEAVWEBI WUQ report which formats work units on a WUQ dispatch queue. For example, you may find an address space has SRBs on a WUQ waiting to be dispatched per IEAVWEBI WUQ, but the IEAVWEBI SRB ASID(xx) report may show no WEBs on the “in flight” queue because the pointer (ASSBSAWQ) to this queue was zero at the time the ASSB was dumped.

The IEAVWEBI SRB ASID(xx) report excerpt shown above shows each WEB having a non-zero WUQ address. The presence of a non-zero WUQ address in a WEB does not imply that the WEB is currently on a WUQ. To determine whether a WEB is on a WUQ, use IEAVWEBI WUQ.
Non-dispatchable SSRBs

Where does SSRBCPSW point?

- IEAVSRBS?
  - PC Suspend
  - Get SSRBLSDP: IP CBF ssrblsdp-120 STR(LSE)
  - LSETARG field should contain 00000317; LSEPSWE points to caller

- IEAVEWAT?
  - PC WAIT
  - Get SSRBLSDP: IP CBF ssrblsdp-120 STR(LSE)
  - LSETARG field should contain 0000030D; LSEPSWE points to caller

- IEAVEPSS?
  - Paused.
  - SSRB Reg13 points to standard save area; +C is return address
Non-dispatchable SSRBs

- Where does SSRBCPSW point? (cont)
  - After a BALR?
    - Could be branch-entered SUSPEND
    - Check code where SSRBCPSW points for SUSPEND/CALLDISP macros
  - Just a “regular instruction”?
    - Could be suspended for a page fault or other translation exception
    - Get SSRXTRNE (SSRX formatted along with SSRB)
    - If non-zero, does it match base register or instruction address of instruction pointed to by SSRBCPSW?
Questions?