CICS Common Performance Problems and Debugging

Ed Addison
IBM

February 6 2013
12440
Agenda

- CICS Dispatcher Basics
- Performance Problem - Loop
- Externalize MXT with CICS System Events
- CICS Monitoring Facility
- RMFIII
- Systrace perfdata
CICS Dispatcher Basics
The box in blue below shows TCBs and SRBs using Logical Processor 1 in an LPAR.

Only one thing (TCB or SRB) can run at a time on this processor.

z/OS decides which TCBs and SRBs run on the processor.

A typical well-behaved Concurrency(quasirent) CICS application program does not usually do anything that would cause the QR TCB to wait or suspend the QR TCB to the z/OS dispatcher.

But, there is nothing to stop this from occurring.
There are several different ways that one TCB or SRB can lose or relinquish control of the processor.

- A TCB can be interrupted while it is executing instructions. Then z/OS can give control of the processor to a higher priority TCB or SRB. The interrupted TCB is left undispatched until z/OS gives it a processor and it can then resume executing instructions.

- A TCB can voluntarily give up control by suspending or waiting to the z/OS dispatcher.
  - That can happen explicitly. For instance, when the CICS dispatcher has no CICS transactions ready to run on the TCB it will issue an SVC 1 wait to temporarily give control back to z/OS so something else can use the processor.

- Paging is another way that a TCB can lose control. If an instructions needs a page of storage that has to be paged in from Aux, then the TCB gives up control of the processor and waits for the I/O. In the interim, the z/OS dispatcher can find another TCB or SRB who wants to run on the processor.
On the QR TCB, CICS has built its own dispatching environment. In that environment CICS transactions share the QR TCB.

The CICS Dispatcher decides which transactions run on the QR TCB.

Here, 4 transactions have all just been attached and are all ready to run on the QR TCB. They are all Dispatchable as indicated by the light red shading.

The CICS Dispatcher picks one and gives it control.
• Tran 10001 received control. It held control of the TCB for 7 units of wall-clock time and then it suspended giving control back to the CICS dispatcher.

• When the CICS dispatcher gets control, it knows how long transaction 10001 had control of the QR TCB, but it doesn’t know how much CPU it used until it asks z/OS with a TIMEUSED. We’ll say it used 7 units of CPU too. That is indicated by the solid green shading in the box.

• The dispatcher now chooses 1 of the 3 dispatchable tasks to dispatch.
• Transaction 10002 had control of the TCB for 13 units of time. But this time it used only 7 units of CPU. So while transaction 10002 was in control of the QR TCB, the QR TCB stopped executing instructions for some reason. Maybe z/OS took control away to let higher priority work use the processor.

• Transaction 10001 is still suspended. Transaction 10002 just suspended. 10003 and 10004 are both dispatchable. The CICS dispatcher picks one of them.
• Tran 10003 ran solidly and suspended.

• Then while in the CICS dispatcher code, control of the QR TCB was taken away for about 7 units of time.

• During that time, at timeline 38, transaction 10002 became dispatchable. That means that whatever it was suspended on has completed. An example is when suspended for File I/O. When the I/O completes and the ECB is posted, the waiting CICS transaction immediately becomes dispatchable.
Here, several transactions ran, each time giving control back to the CICS dispatcher.

Transactions 10001 and 10002 have finished.

At this point, control is in the CICS dispatcher but there are no dispatchable tasks. When that happens, the CICS dispatcher issues an SVC 1 Wait to give control of the TCB back to z/OS temporarily.
During that time, notice 2 transactions became dispatchable. Typically the TCB would wake up out of its wait immediately when a transaction becomes dispatchable. The reason that didn’t happen is probably because the processor was not available.
Here the last 2 tasks finish up. Transaction 10004 lost control of the TCB for some reason during its last dispatch. Since there are no more transactions, the CICS Dispatcher issues another SVC1 Wait.
Problem One: Loop
Problem One - Loop

- Customer called the Support Center for no transactions running yet high CPU consumed by the CICS region

- **ST SYS** (Status SYStem)
  - With a dump, from the IPCS Commands panel, enter the **ST SYS** command to find out what time the dump was taken. Below is an example of the output

  ```
  SYSTEM STATUS:
  Nucleus member name: IEANUC01
  Sysplex name: EDZPLEX
  TIME OF DAY CLOCK: C2D443EE B58366C4 08/11/2008 18:31:27.786295 local
  Program Producing Dump: SVCDUMP
  Program Requesting Dump: IEAVTSDT
  Incident token: EDZPLEX 07/08/2008 23:31:27.198911 GMT
  ```

- When getting information from this screen, it is important to note both the LOCAL Time-Of-Day and the GMT Time-Of-Day
  - The CICS Dispatcher summary gives times as GMT rather than LOCAL
In this example, the CSA time is 16:24:26.8 LOCAL Time

There is other important information given in this output:

- The jobname of CICS
- The address space ID (ASID) of CICS
Compare Times

- Once you have the CSA time and the LOCAL time from ST SYS, you can decide if CICS is healthy or not
  - If there is a several minute time gap between the two times, then you know CICS is unhealthy
  - If the two times are close, then you know CICS is healthy and something else is causing the problem

- Even when a healthy CICS is dumped, there is usually some difference between the two times. This is because ST SYS is not the exact time CICS started dumping
  - If there is less than a minute difference between CSA time and SY SYS LOCAL time, then you can generally say CICS was healthy when the dump was taken

- From this example, the CSA has not been updated in over 2 hours:
  - CSA time is: 16:24:26.8 local
  - ST SYS time is: 18:31:27.7 local

- If the difference between CSA time and ST SYS local time leads you to believe CICS is unhealthy, then this should coincide with CICS CPU utilization
  - When CICS is unhealthy, it is either getting no CPU time (hung) or it is getting all the CPU time (looping)
Is CICS looping or is it hung?

- In this example CICS is not healthy. This indicates the CICS Dispatcher is not getting control for one of 2 reasons:
  - The CICS Dispatcher has given control to a CICS task, and the CICS task has never given control back
  - The CICS Dispatcher has given up control to z/OS, and z/OS has never redispached CICS

- To determine which one it is, enter `VERBX DFHPDxxx ‘KE=1’`
### KE: Kernel Domain KE_TASK Summary

<table>
<thead>
<tr>
<th>KE_NUM</th>
<th>KE_TASK</th>
<th>STATUS</th>
<th>TCA_ADDR</th>
<th>TRAN_#</th>
<th>TRANSID</th>
<th>DS_TASK</th>
<th>KE_KTCB</th>
<th>ERROR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>0EC54C80</td>
<td>KTCB Step</td>
<td>00000000</td>
<td></td>
<td></td>
<td>00000000</td>
<td>0EC96080</td>
<td></td>
</tr>
<tr>
<td>0002</td>
<td>0EC54900</td>
<td>KTCB QR</td>
<td>00000000</td>
<td>10203030</td>
<td>0EC99020</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0003</td>
<td>0EC54580</td>
<td>KTCB RO</td>
<td>00000000</td>
<td>10203148</td>
<td>0EC98040</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0004</td>
<td>0EC54200</td>
<td>KTCB CO</td>
<td>00000000</td>
<td>10203260</td>
<td>1012B020</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0005</td>
<td>0EC71C80</td>
<td>KTCB FO</td>
<td>00000000</td>
<td>10203378</td>
<td>0EC97060</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0006</td>
<td>0EC71900</td>
<td>Not Running</td>
<td>00000000</td>
<td></td>
<td>10136080</td>
<td>0EC98040</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0007</td>
<td>0EC71580</td>
<td>Unused</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0008</td>
<td>0EC71200</td>
<td>KTCB SL</td>
<td>00000000</td>
<td>10203580</td>
<td>10169020</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0009</td>
<td>0EC8EC80</td>
<td>Not Running</td>
<td>00000000</td>
<td>101F3680</td>
<td>0EC99020</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000A</td>
<td>1026E400</td>
<td>KTCB CQ</td>
<td>00000000</td>
<td>10203490</td>
<td>10146020</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0024</td>
<td>101EB900</td>
<td>*<strong>Running</strong></td>
<td>00000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>01A4</td>
<td>116C6080</td>
<td>*<strong>Running</strong></td>
<td>102C3680</td>
<td>84551</td>
<td>CSPG</td>
<td>101CD580</td>
<td>0EC99020</td>
<td></td>
</tr>
</tbody>
</table>
Look to see if there is a **Running** task under the QR TCB. If there is, then the CICS Dispatcher has given control to the task, and the task has not given control back.

- You first need to find the address of the QR KTCB. It is in the KE_KTCB column on the line showing the KTCB QR in the STATUS column.
- In the previous example, you can see there is a running task dispatched on the QR TCB.
- Note: There is another running task, dispatched on the CQ TCB. This is the console/KILL task which remains available for console requests or requests to KILL a looping or hung task.

The task on the QR TCB is ‘running’ from the CICS Dispatcher’s perspective. This simply means it has never given control back to the CICS Dispatcher.

- It could be looping.
- It could have done something causing the CICS QR TCB to lose control.

Find out by using the z/OS System trace and the CICS trace.

Before you look at the z/OS System trace, you need to know the ASID of CICS, and the TCB address of the QR TCB.
Find the address of the QR TCB

- Find the address of the QR TCB by listing the contents of the QR KTCB
  - **IP L 0EC99020 ASID(x‘148’) L(999)**
  - *We obtained this address on slide 18*
  - Offset x’50’ into a KTCB is the address of the corresponding z/OS TCB:

```
LIST 0EC99020. ASID(X’0148’) LENGTH(X’03E7’) AREA
ASID(X’0148’) ADDRESS(0EC99020.) KEY(80)
0EC99020. D2E3C3C2 40404040 00000000 0EC54900 116C6080 0EC5C020 00000159 6D263B20 |KTCB .....E..E{.....__| |0EC99040. 00000000 7D000000 00000000 00000000 80000004 00000000 36800000 D800D8D9 |......’.......................Q.QR|
0EC99060. 00000000 00000000 0EC99020 40000000 00AEB5D8 00000000 00006120 00000000 |........I. ..........| |0EC99080. 00000000 00000000 00000000 00000000 00000000 00000000 00000520 00000038E 00000008 |......................| |0EC990A0. 00AEB5D8 00000000 1012B020 0EC98040 00000000 00000000 00000000 00000000 00000000 |........I...........| |0EC990A0. 00AEB5D8 00000000 1012B020 0EC98040 00000000 00000000 00000000 00000000 00000000 |........I...........| |0EC990A0. 00AEB5D8 00000000 1012B020 0EC98040 00000000 00000000 00000000 00000000 00000000 |........I...........| |OEC990A0. 00AEB5D8 00000000 1012B020 0EC98040 00000000 00000000 00000000 00000000 00000000 |........I...........|
```

- An alternate way to identify the QR TCB is to format the CICS trace table.

  Enter VERBX DFHPDxxx ‘TR=2’, then do a FIND on QR

```
AP 4D01 CQCQ EXIT - FUNCTION(MERGE_CIB_QUEUES) RESPONSE(OK)

  TASK-TCP KE_NUM-001C TCB-QR /00AEB5D8 RET-905E1C0A TIME-16:20:56.9458823764
```

- Now see what the z/OS System trace indicates
### SYSTRACE TIME(LOCAL)

```
PR  ASID  WU-ADDR-  IDENT  CD/D  PSW-----  ADDRESS-  ...  PASD  SASD  TIMESTAMP-LOCAL  CP
06  0148  00AEB5D8  EXT   1005  078D0000  929888E2  ...  0148  0148  18:31:25.338251  01
06  0148  00AEB5D8  I/O    00458  078D2000  929888E6  ...  0148  0148  18:31:25.338279  01
06  0148  00AEB5D8  EXT   1005  078D0000  929888EE  ...  0148  0148  18:31:25.338675  01
06  0148  00AEB5D8  I/O    034D4  078D2000  929888EA  ...  0148  0148  18:31:25.338763  01
06  0148  00AEB5D8  EXT   1005  078D0000  929888EE  ...  0148  0148  18:31:25.339097  01
06  0148  00AEB5D8  I/O    03DF2  078D0000  929888E2  ...  0148  0148  18:31:25.339261  01
06  0148  00AEB5D8  EXT   1005  078D0000  929888EE  ...  0148  0148  18:31:25.339519  01
06  0148  00AEB5D8  CLKC    078D0000  929888E2  ...  0148  0148  18:31:25.339861  01
06  0148  00AEB5D8  DSP   078D0000  929888E2  ...  0148  0148  18:31:25.340108  01
06  0148  00AEB5D8  I/O    0045A  078D2000  929888E6  ...  0148  0148  18:31:25.340167  01
06  0148  00AEB5D8  I/O    00458  078D2000  929888EA  ...  0148  0148  18:31:25.340264  01
06  0148  00AEB5D8  EXT   1005  078D0000  929888E2  ...  0148  0148  18:31:25.340535  01
06  0148  00AEB5D8  I/O    03DF2  078D0000  929888E2  ...  0148  0148  18:31:25.340787  01
06  0148  00AEB5D8  EXT   1005  078D2000  929888EA  ...  0148  0148  18:31:25.340957  01
06  0148  00AEB5D8  I/O    03DF2  078D2000  929888EA  ...  0148  0148  18:31:25.341303  01
06  0148  00AEB5D8  EXT   1005  078D0000  929888EE  ...  0148  0148  18:31:25.341380  01
03  0148  00AEB5D8  DSP   078D0000  929888EE  ...  0148  0148  18:31:25.341420  04
```

- Note: 00AEB5D8 is the QR TCB derived from the previous slide

---

**SHARE San Francisco - Ed Addison**
System Trace Table (cont)

- Verify the ASID being traced is the one for the CICS region we care about
  - If it isn’t, you can enter `SYSTRACE TIME(LOCAL) ASID(x’xx’)`

- The TCB address shows up in the second column. Verify it is the QR TCB

- This trace shows a loop on the QR TCB. Notice the PSW address on the DSP and EXT trace entries

- Before we saw this trace, we already knew CICS was unhealthy
  - A CICS task had not relinquished control to the CICS Dispatcher for over two hours

- By looking at the System trace table, we can verify if the CICS task was looping, or if it had lost control to z/OS
  - Since we see trace entries for the QR TCB, we assume it is looping
CICS is Looping

- What you expect to see in the System trace table is a looping pattern. In this example, we have a pattern of DSP and EXT trace entries
  - EXT trace entries are an external interrupt
    - z/OS is taking control away from the TCB in order to process some sort of interrupt (an I/O interrupt in this case)
  - DSP trace entries are z/OS Dispatcher trace entries
    - The z/OS Dispatcher is giving control back to the TCB at the exact instruction address where control was taken
  - I/O trace entries are z/OS high priority interrupts when I/O finishes
  - CLKC trace entries are z/OS checking clocks when z/OS services haven’t been requested for awhile

- By looking at the PSW addresses in the System trace, you can begin to learn what module(s) comprise the loop
  - If there are several modules involved in the loop, you would likely need to look at lots of I/O, EXT, DSP entries before you could get a handle on the extent of the loop
  - In this example, it is clear fairly quickly the problem is a tight loop involving only a few instructions between address 129888E2 and 129888EE
Finding the Looping Program

- The next step is to identify the program(s) in which the looping instructions live.
- If you are in SYSTRACE, and want to know what module a PSW address falls within, you first need to subtract the high-order bit (the x’80’ bit, if there is one).
  - For instance, if the PSW address is 81234568, then the address you need to use is 1234568.
  - If the PSW address is A1234568, then the address you need to use will be 21234568.

- Once you have the address aaaaaaaaa, you have several choices for figuring out the module:
  - VERBX DFHPDxxx ‘LD=1’ displays the Loader Domain summary information.
    - Enter FIND ‘PROGRAM STORAGE MAP’
    - The Program Storage Map lists the modules loaded by CICS, in address order.
    - In our example, for PSW address 929888E2, we could do a FIND on ‘129’ to get closer to the programs listed near this address.
VERBX DFHPDxxx ‘LD=1’

---

**PUBLIC RD**

**PRP: IBM Software Group**

**VERBX DFHPDxxx ‘LD=1’**

---

**==LD: PROGRAM STORAGE MAP**

<table>
<thead>
<tr>
<th>PGM NAME</th>
<th>ENTRY PT</th>
<th>CSECT</th>
<th>LOAD PT.</th>
<th>REL.</th>
<th>PTF</th>
<th>LVL.</th>
<th>LAST COMPILED</th>
<th>COPY NO.</th>
<th>USERS</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFHCSA</td>
<td>8004E200</td>
<td>DFHKELCL</td>
<td>0004D000</td>
<td>650</td>
<td>HCI6700</td>
<td>06/05/11 05.51</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>-noheda-</td>
<td>0004D4F8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DFHKELRT</td>
<td>0004D900</td>
<td>DFHCSAOF</td>
<td>0004D4F8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>-noheda-</td>
<td>0004D8F8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DFHCSAOF</td>
<td>0004E000</td>
<td>DFHCSA</td>
<td>0004D900</td>
<td>650</td>
<td>HCI6700</td>
<td>05/11 06.53</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0004D8F8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DFHKERCD</td>
<td>0004E4B0</td>
<td>DFHCSA</td>
<td>0004E4B0</td>
<td>650</td>
<td>HCI6700</td>
<td>06/05/11 05.51</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

---

**... then FIND on ‘ 129’ shows:**

<table>
<thead>
<tr>
<th>PGM NAME</th>
<th>ENTRY PT</th>
<th>CSECT</th>
<th>LOAD PT.</th>
<th>REL.</th>
<th>PTF</th>
<th>LVL.</th>
<th>LAST COMPILED</th>
<th>COPY NO.</th>
<th>USERS</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFHCRS</td>
<td>92982D70</td>
<td>DFHCRS</td>
<td>12982D50</td>
<td>650</td>
<td>HCI6700</td>
<td>29/11 23.23</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>DFHSNP</td>
<td>92984BE8</td>
<td>DFHYA630</td>
<td>12984BC0</td>
<td>630</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>12984BC0</td>
<td>630</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>DFHTPR</td>
<td>92987FFA</td>
<td>DFHTPR</td>
<td>12987FD0</td>
<td>650</td>
<td>HCI6700</td>
<td>13/12 11.09</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>GGGGPEND</td>
<td>9298D538</td>
<td>DFHYA640</td>
<td>1298D510</td>
<td>640</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Using the WHERE command or Browse mode

- Enter **WHERE aaaaaaaa** or simply **W aaaaaaaa** from a command line
  - If the first digit of the address starts with a letter, then you could enter the WHERE command followed by a period:
    - **WHERE aaaaaaa.**  e.g. **WHERE C00498.**
  - Or you could include a leading zero:
    - **WHERE 0aaaaaaa**  e.g. **WHERE 0C00498**

- The WHERE command is useful when CICS doesn’t know about the module
  - WHERE is not helpful in this example because the looping module was loaded by CICS, not z/OS, so z/OS is unable to identify the module. **WHERE 129888E2** displays:

  ```
  ASID(X'0148') 129888E2. AREA(Subpool252Key00)+1888E2 IN EXTENDED PRIVATE
  ```

- You can also try to display the PSW address in Browse mode and back up looking for an eyecatcher
The CICS Trace Table

- If the System trace entries indicate the loop is larger than a tight loop within one module, it is possible CICS services are being requested by the looping module. If this is true and if CICS internal trace is active, then you may be able to see the loop in the CICS trace.

**Note:** Some CICS services (like EXEC CICS SUSPEND or EXEC CICS SEND WAIT) cause the task to be suspended and the CSA Time-of-Day clock to be updated. Because we have determined CICS is unhealthy in this discussion, we know no such services are being requested. Other CICS services (like EXEC CICS ASSIGN or EXEC CICS FREEMAIN) do not cause the task to be suspended (i.e. do not give control back to the CICS Dispatcher).

- To format the internal CICS trace table, enter `VERBX DFHPDxxx ‘TR=2’`
The trace entries below are the last trace entries in the dump. The time stamps match the CSA Time-of-Day. This is consistent with a tight loop. As soon as the tight loop starts, there are no more CICS trace entries, no more updates of the CSA Time-of-Day, and no more useful work done by CICS.

```
AP 1940 APLI ENTRY - FUNCTION(START_PROGRAM) PROGRAM(DFHTPR) CEDF_STATUS(NOCEDF) EXECUTION_SET(FULLAPI)
SYNCONRETURN(NO) LANGUAGE_BLOCK(1174EAE0) COMMAREA(00000000 , 00000000) LINK_LEVEL(1)
    TASK-84551 KE_NUM-01A4 TCB-QR /00AEB5D8 RET-9047C94E TIME-16:24:26.8881611423
XM 1001 XMIQ ENTRY - FUNCTION(INQUIRE_TRANSACTION)
    TASK-84551 KE_NUM-01A4 TCB-QR /00AEB5D8 RET-9059A64C TIME-16:24:26.8881644079
XM 1002 XMIQ EXIT - FUNCTION(INQUIRE_TRANSACTION) RESPONSE(OK) FACILITY_TYPE(TERMINAL) TRANNUM(0084551C)
    ORIGINAL_TRANSACTION_ID(CSPG)
    TASK-84551 KE_NUM-01A4 TCB-QR /00AEB5D8 RET-9059A64C TIME-16:24:26.8881673220
---TRACE TABLE END---
```

NOTE: Time to call the Support Center for loop in DFH module
Externalize CICS MaxTask with System Event
Externalize MXT with System Events

- New with CICS Transaction Server 4.2
- Event processing supports the following system events:
  - FILE enable or disable status
  - FILE open or close status
  - DB2CONN connection status
  - TASK threshold
  - TRANCLASS TASK threshold
  - Unhandled transaction abend
- Use CICS Explorer to build Event Binding file
- Prepare and install a Transaction and a Program that will write out a console message at various task thresholds.
- Set a SLIP to get a dump on one of the messages.
Problem: IYNXK went MaXTask

07.10.08 JOB18137  +ABOVE_60_PERCENT_OF_MXT
07.10.14 JOB18137  +ABOVE_80_PERCENT_OF_MXT
07.10.17 JOB18137  IEA794I SVC DUMP HAS CAPTURED: 032
032  DUMPID=154 REQUESTED BY JOB (IYNXK )
032  DUMP TITLE=SLIP DUMP ID=AB80
07.10.20 JOB18137  +ABOVE_100_PERCENT_OF_MXT
07.10.24 JOB18137  +BELOW_90_PERCENT_OF_MXT
07.10.26 JOB18137  +ABOVE_100_PERCENT_OF_MXT
07.10.36 JOB18137  +BELOW_90_PERCENT_OF_MXT
07.10.38 JOB18137  +BELOW_70_PERCENT_OF_MXT
07.10.45 JOB18137  +ABOVE_80_PERCENT_OF_MXT
07.10.51 JOB18137  +BELOW_70_PERCENT_OF_MXT
07.10.53 JOB18137  +BELOW_50_PERCENT_OF_MXT
07.11.00 JOB18137  IEA794I SVC DUMP HAS CAPTURED: 073
073  DUMPID=155 REQUESTED BY JOB (*MASTER*)
073  DUMP TITLE=IYNXK MXT
07.11.18 JOB18137  +ABOVE_60_PERCENT_OF_MXT
07.11.20 JOB18137  +ABOVE_80_PERCENT_OF_MXT
07.11.22 JOB18137  +ABOVE_100_PERCENT_OF_MXT
07.11.28 JOB18137  +BELOW_90_PERCENT_OF_MXT
07.11.30 JOB18137  +BELOW_70_PERCENT_OF_MXT
07.11.32 JOB18137  +ABOVE_80_PERCENT_OF_MXT
07.11.34 JOB18137  +ABOVE_100_PERCENT_OF_MXT
07.11.46 JOB18137  IEA794I SVC DUMP HAS CAPTURED: 117
117  DUMPID=156 REQUESTED BY JOB (*MASTER*)
117  DUMP TITLE=IYNXK MXT2
07.12.00 JOB18137  +BELOW_90_PERCENT_OF_MXT
07.12.01 JOB18137  +ABOVE_100_PERCENT_OF_MXT
07.12.06 JOB18137  +BELOW_90_PERCENT_OF_MXT
07.12.07 JOB18137  +ABOVE_100_PERCENT_OF_MXT
07.12.12 JOB18137  +ABOVE_100_PERCENT_OF_MXT
07.12.17 JOB18137  +BELOW_90_PERCENT_OF_MXT
07.12.18 JOB18137  +ABOVE_100_PERCENT_OF_MXT
07.12.26 JOB18137  +BELOW_90_PERCENT_OF_MXT
07.12.28 JOB18137  +ABOVE_100_PERCENT_OF_MXT
07.13.08 JOB18137  +BELOW_90_PERCENT_OF_MXT
07.13.10 JOB18137  +BELOW_70_PERCENT_OF_MXT
07.13.11 JOB18137  +BELOW_50_PERCENT_OF_MXT
07.15.27 JOB18137  IEA794I SVC DUMP HAS CAPTURED: 197
197  DUMPID=157 REQUESTED BY JOB (*MASTER*)
197  DUMP TITLE=IYNXK NORM
Create an Event Binding Specification that contains 6 Capture Specifications as shown. The name of each Capture Specification is the content of the message sent to the console.
For each Capture Specification, choose a TASK THRESHOLD System Capture Point
For each Capture Specification, define a predicate that matches the name of the Capture Specification.
For the Adapter, choose Custom (User Written) and put in a Transaction ID. Then click on Advanced Options.
In advanced Options, let everything default except specify Dispatch Priority High.
Translate, Assemble, and Link the following program into a dataset in the DFHRPL concatenation

```
TITLE 'EPADAPTR'
***********************************************************************
* EPADAPTR: Puts out a message to the console                       *
*---------------------------------------------------------------------*
DFHEISTG DSECT
STRUCLEN DS CL4
*
DFHREGS
COPY DFHEPCXD Covers DFHEP.CONTEXT container
COPY DFHEPDED Covers DFHEP.DESCRIPTOR container
COPY DFHEPAPD Covers DFHEP.ADAPTPARM container
*
EPADAPTR CSECT
EPADAPTR AMODE ANY
EPADAPTR RMODE ANY
*
EXEC CICS GET CONTAINER('DFHEP.CONTEXT') X
SET(R9) FLENGTH(STRUCLEN)
USING EPCX,R9
EXEC CICS WRITE OPERATOR TEXT(EPCX_CS_NAME)
EXEC CICS RETURN
*
END
```
Final Steps:

- Export the Bundle Project containing the Event Binding Specification.
- Define and Install a Transaction definition for EPTT and a Program definition for EPADAPTR. Specify Priority(255) on the EPTT transaction definition.
- Using the exported Bundle Project file, define and install the Bundle
- And if you want to get a dump on one of the messages, here is a SLIP:

```
SLIP SET, MSGID='ABOVE_80', J=jobname, ID=AB80, A=SVCD, ML=1, END
```
CICS Monitoring Facility Information

- Two CICS/PA summary forms
- Use them with the 4 example tasks
- Use them with the problem SMF110 data
SUSPSUM summarizes components of Suspend Time.
DISPSUM summarizes components of Dispatch Time.

```
SUMMARY(OUTPUT(SUSPSUM),
  EXTERNAL(CPAXW001),
  TOTALS(8),
  INTERVAL(00:00:30),
  FIELDS(START(TIMES,ASCEND),
    TASKCNT,
    RESPONSE(AVE),
    DISPATCH(TIME(AVE)),
    SUSPEND(TIME(AVE)),
    SUSPEND(COUNT(AVE)),
    DSPDELAY(TIME(AVE)),
    MXTDELAY(TIME(AVE)),
    TCLDELAY(TIME(AVE)),
    DISPWAIT(TIME(AVE)),
    QRMODDLY(TIME(AVE)),
    FCPWAIT(TIME(AVE)),
    FCPWAIT(COUNT(AVE)))))
```

```
SUMMARY(OUTPUT(DISPSUM),
  EXTERNAL(CPAXW002),
  TOTALS(8),
  INTERVAL(00:00:30),
  FIELDS(START(TIMES,ASCEND),
    TASKCNT,
    RESPONSE(AVE),
    SUSPEND(TIME(AVE)),
    DISPATCH(TIME(AVE)),
    CPU(TIME(AVE)),
    QRDISPT(TIME(TOT)),
    QRDISPT(TIME(AVE)),
    QRCPU(TIME(TOT)),
    KY8DISPT(TIME(AVE)),
    KY8DISPT(COUNT(AVE)),
    L8CPU(TIME(AVE)),
    MXTDELAY(TIME(AVE)))))
```
<table>
<thead>
<tr>
<th>Interval</th>
<th>#Tasks</th>
<th>Response Avg Time</th>
<th>Dispatch Avg Time</th>
<th>Suspend Avg Time</th>
<th>Suspend Disp1Dly Avg Count</th>
<th>MXTDelay Avg Time</th>
<th>TCLDelay Avg Time</th>
<th>DispWait Avg Time</th>
<th>QRModDly Avg Time</th>
<th>FC Wait Avg Time</th>
<th>FC Wait Avg Time</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>07:08:30</td>
<td>3228</td>
<td>.1113</td>
<td>.1092</td>
<td>.1021</td>
<td>1</td>
<td>.0605</td>
<td>.0000</td>
<td>.0000</td>
<td>.0414</td>
<td>.0414</td>
<td>.0416</td>
<td>0</td>
</tr>
<tr>
<td>07:09:00</td>
<td>3276</td>
<td>.1562</td>
<td>.0913</td>
<td>.1471</td>
<td>1</td>
<td>.0830</td>
<td>.0000</td>
<td>.0000</td>
<td>.0638</td>
<td>.0638</td>
<td>.0641</td>
<td>0</td>
</tr>
<tr>
<td>07:09:30</td>
<td>3228</td>
<td>.3328</td>
<td>.0913</td>
<td>.3234</td>
<td>1</td>
<td>.1698</td>
<td>.0000</td>
<td>.0000</td>
<td>.1525</td>
<td>.1524</td>
<td>.1528</td>
<td>0</td>
</tr>
<tr>
<td>07:10:00</td>
<td>2285</td>
<td>2.1023</td>
<td>.0137</td>
<td>2.0886</td>
<td>1</td>
<td>1.0377</td>
<td>.0289</td>
<td>.0000</td>
<td>1.0375</td>
<td>1.0375</td>
<td>1.0076</td>
<td>0</td>
</tr>
<tr>
<td>07:10:30</td>
<td>2105</td>
<td>1.5692</td>
<td>.0131</td>
<td>1.5561</td>
<td>1</td>
<td>.7964</td>
<td>.0083</td>
<td>.0000</td>
<td>.7540</td>
<td>.7540</td>
<td>.7083</td>
<td>0</td>
</tr>
<tr>
<td>07:11:00</td>
<td>2388</td>
<td>1.1418</td>
<td>.0125</td>
<td>1.1293</td>
<td>1</td>
<td>.5423</td>
<td>.0434</td>
<td>.0000</td>
<td>.5813</td>
<td>.5813</td>
<td>.5195</td>
<td>0</td>
</tr>
<tr>
<td>07:11:30</td>
<td>1945</td>
<td>3.4445</td>
<td>.0158</td>
<td>3.4287</td>
<td>1</td>
<td>1.8043</td>
<td>.3032</td>
<td>.0000</td>
<td>1.6064</td>
<td>1.6064</td>
<td>1.4462</td>
<td>0</td>
</tr>
<tr>
<td>07:12:00</td>
<td>2446</td>
<td>2.4340</td>
<td>.0117</td>
<td>2.4223</td>
<td>1</td>
<td>1.2851</td>
<td>.1436</td>
<td>.0000</td>
<td>1.1246</td>
<td>1.1246</td>
<td>.9916</td>
<td>0</td>
</tr>
<tr>
<td>07:12:30</td>
<td>3240</td>
<td>1.7993</td>
<td>.0091</td>
<td>1.7902</td>
<td>1</td>
<td>.9038</td>
<td>.0030</td>
<td>.0000</td>
<td>.8778</td>
<td>.8778</td>
<td>.8823</td>
<td>0</td>
</tr>
<tr>
<td>07:13:00</td>
<td>3051</td>
<td>6.163</td>
<td>.0091</td>
<td>6.072</td>
<td>1</td>
<td>.3217</td>
<td>.0000</td>
<td>.0000</td>
<td>.2806</td>
<td>.2806</td>
<td>.2843</td>
<td>0</td>
</tr>
<tr>
<td>07:13:30</td>
<td>3252</td>
<td>.0753</td>
<td>.0091</td>
<td>.0661</td>
<td>1</td>
<td>.0413</td>
<td>.0000</td>
<td>.0000</td>
<td>.0246</td>
<td>.0246</td>
<td>.0248</td>
<td>0</td>
</tr>
<tr>
<td>07:14:00</td>
<td>3258</td>
<td>1.391</td>
<td>.0091</td>
<td>1.300</td>
<td>1</td>
<td>.0742</td>
<td>.0000</td>
<td>.0000</td>
<td>.0556</td>
<td>.0556</td>
<td>.0559</td>
<td>0</td>
</tr>
<tr>
<td>07:14:30</td>
<td>3258</td>
<td>.1640</td>
<td>.0091</td>
<td>.1548</td>
<td>1</td>
<td>.0867</td>
<td>.0000</td>
<td>.0000</td>
<td>.0679</td>
<td>.0679</td>
<td>.0682</td>
<td>0</td>
</tr>
</tbody>
</table>

- Average Response Time started going bad in the 7:09:30 interval.
- It was back to normal starting in the 7:13:30 interval.
- You can see from the MXTDelay column which intervals had some MXT delay.
**Notice that Response time is always Dispatch time plus Suspend time. A task is always either Suspended or Dispatched.**
<table>
<thead>
<tr>
<th>Start</th>
<th>#Tasks</th>
<th>Avg Task</th>
<th>Avg Resp</th>
<th>Avg Disp</th>
<th>Avg Susp</th>
<th>Avg Susp Disp</th>
<th>Avg MXT Delay</th>
<th>Avg TCL Delay</th>
<th>Avg Disp Wait</th>
<th>Avg QR Mod Delay</th>
<th>Avg FC Wait</th>
<th>Avg FC Wait</th>
</tr>
</thead>
<tbody>
<tr>
<td>07:08:30</td>
<td>3228</td>
<td>0.1113</td>
<td>0.0092</td>
<td>0.1021</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>07:09:00</td>
<td>3276</td>
<td>0.1562</td>
<td>0.0091</td>
<td>0.1471</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>07:09:30</td>
<td>3228</td>
<td>0.3328</td>
<td>0.0093</td>
<td>0.3234</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>07:10:00</td>
<td>2285</td>
<td>2.1023</td>
<td>0.0137</td>
<td>2.0886</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>07:10:30</td>
<td>2105</td>
<td>1.5692</td>
<td>0.0131</td>
<td>1.5561</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>07:11:00</td>
<td>2384</td>
<td>1.1418</td>
<td>0.0125</td>
<td>1.1293</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>07:11:30</td>
<td>1945</td>
<td>3.4445</td>
<td>0.0158</td>
<td>3.4287</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>07:12:00</td>
<td>2446</td>
<td>2.4340</td>
<td>0.0117</td>
<td>2.4223</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>07:12:30</td>
<td>3240</td>
<td>1.7993</td>
<td>0.0091</td>
<td>1.7902</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>07:13:00</td>
<td>3051</td>
<td>0.6163</td>
<td>0.0091</td>
<td>0.6072</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>07:13:30</td>
<td>3252</td>
<td>0.0753</td>
<td>0.0091</td>
<td>0.0661</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>07:14:00</td>
<td>3258</td>
<td>0.1391</td>
<td>0.0091</td>
<td>0.1300</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>07:14:30</td>
<td>3258</td>
<td>0.1640</td>
<td>0.0091</td>
<td>0.1548</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Get used to what is normal. Dispatch time is normally about .0091. That increases significantly during the problem intervals. Suspend time is normally about .1300. That increases significantly during the problem intervals.
Let’s graph the 07:08:30 30-second interval. It is a normal, pre-problem interval.

Disp1Dly is 60 milliseconds and there is no MXTDelay or TCLDelay. So all 60 milliseconds is dispatchable, waiting to run on the QR.
### SUSPSUM

<table>
<thead>
<tr>
<th>Start</th>
<th>#Tasks</th>
<th>Response</th>
<th>Dispatch</th>
<th>Suspend</th>
<th>Sus pend</th>
<th>Disp1Dly</th>
<th>MXTDelay</th>
<th>TCLDelay</th>
<th>DispWait</th>
<th>QRModDly</th>
<th>FC Wait</th>
<th>FC Wait</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>07:08:30</td>
<td>3228</td>
<td>.1113</td>
<td>.0092</td>
<td>.1021</td>
<td>1</td>
<td>.0605</td>
<td>.0000</td>
<td>.0000</td>
<td>.0414</td>
<td>.0414</td>
<td>.0416</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

### DISPSUM

<table>
<thead>
<tr>
<th>Start</th>
<th>#Tasks</th>
<th>Response</th>
<th>Dispatch</th>
<th>User</th>
<th>CPU</th>
<th>QR Disp</th>
<th>QR Disp</th>
<th>QR CPU</th>
<th>KY8 Disp</th>
<th>KY8 Disp</th>
<th>L8 CPU</th>
<th>MXTDelay</th>
</tr>
</thead>
<tbody>
<tr>
<td>07:08:30</td>
<td>3228</td>
<td>.1113</td>
<td>.1021</td>
<td>.0086</td>
<td>29.6445</td>
<td>.0092</td>
<td>.0086</td>
<td>.0000</td>
<td>0</td>
<td>.0000</td>
<td>.0000</td>
<td></td>
</tr>
</tbody>
</table>

- Suspend Time is 102 milliseconds and Disp1Dly is 60 milliseconds. So the remaining part of Suspend time is 42 milliseconds. Of that, 41 milliseconds is waiting for redispatch (DispWait) on the QR (QRModDly).

- So, almost the whole 102 millisecond suspend time is waiting to run on the QR. Clearly the QR TCB is a bottleneck, during normal intervals.
**Dispatch Time is 9 milliseconds.** Notice that QR Disp is the same. So we know that the transactions only ran on the QR TCB.

**User CPU (and QR CPU) round up to 9 milliseconds.** So we’ll make the whole 9 milliseconds dark green.

![Diagram showing the breakdown of task response times](image-url)
Here is the DISPSUM form showing dispatch time fields.

Notice that Dispatch Time and QR Disp Time are the same. That means that all processing is on the QR TCB.
<table>
<thead>
<tr>
<th>Start Interval</th>
<th>#Tasks</th>
<th>Response Time</th>
<th>Suspend Time</th>
<th>Dispatch Time</th>
<th>User CPU Time</th>
<th>QR Disp Time</th>
<th>QR CPU Time</th>
<th>Disp KY8 Time</th>
<th>KY8 Disp Time</th>
<th>L8 CPU MXT Delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>07:08:30</td>
<td>3228</td>
<td>.1113</td>
<td>.1021</td>
<td>.0092</td>
<td>.0086</td>
<td>29.6445</td>
<td>.0092</td>
<td>.0086</td>
<td>.0000</td>
<td>0</td>
</tr>
<tr>
<td>07:09:00</td>
<td>3276</td>
<td>.1562</td>
<td>.1471</td>
<td>.0091</td>
<td>.0086</td>
<td>29.6774</td>
<td>.0091</td>
<td>.0086</td>
<td>.0000</td>
<td>0</td>
</tr>
<tr>
<td>07:09:30</td>
<td>3282</td>
<td>.3328</td>
<td>.3234</td>
<td>.0093</td>
<td>.0088</td>
<td>30.1325</td>
<td>.0093</td>
<td>.0088</td>
<td>.0000</td>
<td>0</td>
</tr>
<tr>
<td>07:10:00</td>
<td>2285</td>
<td>2.1023</td>
<td>2.0886</td>
<td>.0137</td>
<td>.0115</td>
<td>31.3524</td>
<td>.0137</td>
<td>.0115</td>
<td>.0000</td>
<td>0</td>
</tr>
<tr>
<td>07:10:30</td>
<td>2105</td>
<td>1.5692</td>
<td>1.5561</td>
<td>.0131</td>
<td>.0115</td>
<td>27.4879</td>
<td>.0131</td>
<td>.0115</td>
<td>.0000</td>
<td>0</td>
</tr>
<tr>
<td>07:11:00</td>
<td>2384</td>
<td>1.1418</td>
<td>1.1293</td>
<td>.0125</td>
<td>.0115</td>
<td>29.8614</td>
<td>.0125</td>
<td>.0115</td>
<td>.0000</td>
<td>0</td>
</tr>
<tr>
<td>07:11:30</td>
<td>1945</td>
<td>3.4445</td>
<td>3.4287</td>
<td>.0158</td>
<td>.0117</td>
<td>30.8260</td>
<td>.0158</td>
<td>.0117</td>
<td>.0000</td>
<td>0</td>
</tr>
<tr>
<td>07:12:00</td>
<td>2446</td>
<td>2.4340</td>
<td>2.4223</td>
<td>.0117</td>
<td>.0106</td>
<td>28.6731</td>
<td>.0117</td>
<td>.0106</td>
<td>.0000</td>
<td>0</td>
</tr>
<tr>
<td>07:12:30</td>
<td>3204</td>
<td>1.7993</td>
<td>1.7902</td>
<td>.0091</td>
<td>.0086</td>
<td>29.5015</td>
<td>.0091</td>
<td>.0086</td>
<td>.0000</td>
<td>0</td>
</tr>
<tr>
<td>07:13:00</td>
<td>3051</td>
<td>.6163</td>
<td>.6072</td>
<td>.0091</td>
<td>.0086</td>
<td>27.8417</td>
<td>.0091</td>
<td>.0086</td>
<td>.0000</td>
<td>0</td>
</tr>
<tr>
<td>07:13:30</td>
<td>3252</td>
<td>.0753</td>
<td>.0661</td>
<td>.0091</td>
<td>.0086</td>
<td>29.7519</td>
<td>.0091</td>
<td>.0086</td>
<td>.0000</td>
<td>0</td>
</tr>
<tr>
<td>07:14:00</td>
<td>3258</td>
<td>.1391</td>
<td>.1300</td>
<td>.0091</td>
<td>.0086</td>
<td>29.6127</td>
<td>.0091</td>
<td>.0086</td>
<td>.0000</td>
<td>0</td>
</tr>
<tr>
<td>07:14:30</td>
<td>3258</td>
<td>.1640</td>
<td>.1548</td>
<td>.0091</td>
<td>.0086</td>
<td>29.6975</td>
<td>.0091</td>
<td>.0086</td>
<td>.0000</td>
<td>0</td>
</tr>
</tbody>
</table>

- This chart summarizes all the tasks that started during each 30-second interval. Notice how, even during the good intervals, Total QR Disp time is very close to 30 seconds. This is further evidence that the QR TCB is a bottleneck. That squares with how almost all of the Suspend time is waiting to run on the QR TCB.
### DISPSUM

<table>
<thead>
<tr>
<th>Start Interval</th>
<th>#Tasks</th>
<th>Response Time Avg</th>
<th>Suspend Time Avg</th>
<th>Dispatch Time Avg</th>
<th>User CPU Time Avg</th>
<th>QR Disp Time Avg</th>
<th>Total Time Avg</th>
<th>QR Disp Count</th>
<th>QR CPU Disp Time Avg</th>
<th>KY8 Disp Time Avg</th>
<th>L8 CPU MXT Delay Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>07:08:30</td>
<td>3228</td>
<td>.1113</td>
<td>.1021</td>
<td>.0092</td>
<td>.0086</td>
<td>29.6445</td>
<td>.0092</td>
<td>.0086</td>
<td>0.0000</td>
<td>0.0000</td>
<td></td>
</tr>
<tr>
<td>07:09:00</td>
<td>3276</td>
<td>.1562</td>
<td>.1471</td>
<td>.0091</td>
<td>.0086</td>
<td>29.6774</td>
<td>.0091</td>
<td>.0086</td>
<td>0.0000</td>
<td>0.0000</td>
<td></td>
</tr>
<tr>
<td>07:09:30</td>
<td>3228</td>
<td>.3328</td>
<td>.3234</td>
<td>.0093</td>
<td>.0088</td>
<td>30.1325</td>
<td>.0093</td>
<td>.0088</td>
<td>0.0000</td>
<td>0.0000</td>
<td></td>
</tr>
<tr>
<td><strong>07:10:00</strong></td>
<td>2285</td>
<td>2.1023</td>
<td>2.0886</td>
<td>.0137</td>
<td>.0115</td>
<td><strong>31.3524</strong></td>
<td>.0137</td>
<td>.0115</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0289</td>
</tr>
<tr>
<td>07:10:30</td>
<td>2105</td>
<td>1.5692</td>
<td>1.5561</td>
<td>.0131</td>
<td>.0115</td>
<td>27.4879</td>
<td>.0131</td>
<td>.0115</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0083</td>
</tr>
<tr>
<td>07:11:00</td>
<td>2384</td>
<td>1.1418</td>
<td>1.1293</td>
<td>.0125</td>
<td>.0115</td>
<td>29.8614</td>
<td>.0125</td>
<td>.0115</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0434</td>
</tr>
<tr>
<td>07:11:30</td>
<td>1945</td>
<td>3.4445</td>
<td>3.4287</td>
<td>.0158</td>
<td>.0117</td>
<td>30.8260</td>
<td>.0158</td>
<td>.0117</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.3032</td>
</tr>
<tr>
<td>07:12:00</td>
<td>2446</td>
<td>2.4340</td>
<td>2.4223</td>
<td>.0117</td>
<td>.0106</td>
<td>28.6731</td>
<td>.0117</td>
<td>.0106</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.1436</td>
</tr>
<tr>
<td>07:12:30</td>
<td>3240</td>
<td>1.7993</td>
<td>1.7902</td>
<td>.0091</td>
<td>.0086</td>
<td>29.5015</td>
<td>.0091</td>
<td>.0086</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0030</td>
</tr>
<tr>
<td>07:13:00</td>
<td>3051</td>
<td>.6163</td>
<td>.6072</td>
<td>.0091</td>
<td>.0086</td>
<td>27.8617</td>
<td>.0091</td>
<td>.0086</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>07:13:30</td>
<td>3252</td>
<td>.0753</td>
<td>.0661</td>
<td>.0091</td>
<td>.0086</td>
<td>29.7519</td>
<td>.0091</td>
<td>.0086</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>07:14:00</td>
<td>3258</td>
<td>.1391</td>
<td>.1300</td>
<td>.0091</td>
<td>.0086</td>
<td>29.6127</td>
<td>.0091</td>
<td>.0086</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>07:14:30</td>
<td>3258</td>
<td>.1640</td>
<td>.1548</td>
<td>.0091</td>
<td>.0086</td>
<td>29.6975</td>
<td>.0091</td>
<td>.0086</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
</tbody>
</table>

• How is it possible for tasks that ran in a 30-second interval to use more than 30 seconds of QR Disp time? It is because these intervals include all tasks that started within the interval. For example, tasks that started at 07:10:29.9 are included in the 07:10:00 interval even though all of their processing is after 07:10:30.
<table>
<thead>
<tr>
<th>DISPSUM</th>
<th>Avg</th>
<th>Avg</th>
<th>Avg</th>
<th>Avg</th>
<th>Total</th>
<th>Avg</th>
<th>Avg</th>
<th>Avg</th>
<th>Avg</th>
<th>Avg</th>
<th>Avg</th>
<th>Avg</th>
<th>Avg</th>
<th>Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>#Tasks</td>
<td>Response</td>
<td>Suspend</td>
<td>Dispatch</td>
<td>User</td>
<td>CPU</td>
<td>QR Disp</td>
<td>QR Disp</td>
<td>QR CPU</td>
<td>KY8 Disp</td>
<td>KY8 Disp</td>
<td>L8 CPU</td>
<td>MXTDelay</td>
<td></td>
</tr>
<tr>
<td>Interval</td>
<td>Time</td>
<td>Time</td>
<td>Time</td>
<td>Time</td>
<td>Time</td>
<td>Time</td>
<td>Time</td>
<td>Time</td>
<td>Time</td>
<td>Time</td>
<td>Time</td>
<td>Time</td>
<td>Time</td>
<td></td>
</tr>
<tr>
<td>07:08:30</td>
<td>3228</td>
<td>.1113</td>
<td>.1021</td>
<td>.0092</td>
<td>.0086</td>
<td>29.6445</td>
<td>.0092</td>
<td>.0086</td>
<td>.0000</td>
<td>0</td>
<td>.0000</td>
<td>.0000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>07:09:00</td>
<td>3276</td>
<td>.1562</td>
<td>.1471</td>
<td>.0091</td>
<td>.0086</td>
<td>29.6774</td>
<td>.0091</td>
<td>.0086</td>
<td>.0000</td>
<td>0</td>
<td>.0000</td>
<td>.0000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>07:09:30</td>
<td>3228</td>
<td>.3328</td>
<td>.3234</td>
<td>.0093</td>
<td>.0088</td>
<td>30.1325</td>
<td>.0093</td>
<td>.0088</td>
<td>.0000</td>
<td>0</td>
<td>.0000</td>
<td>.0000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>07:10:00</td>
<td>2285</td>
<td>2.1023</td>
<td>2.0886</td>
<td>.0137</td>
<td>.0115</td>
<td>31.3524</td>
<td>.0137</td>
<td>.0115</td>
<td>.0000</td>
<td>0</td>
<td>.0000</td>
<td>.0289</td>
<td></td>
<td></td>
</tr>
<tr>
<td>07:10:30</td>
<td>2105</td>
<td>1.5692</td>
<td>1.5561</td>
<td>.0131</td>
<td>.0115</td>
<td>27.4879</td>
<td>.0131</td>
<td>.0115</td>
<td>.0000</td>
<td>0</td>
<td>.0000</td>
<td>.0083</td>
<td></td>
<td></td>
</tr>
<tr>
<td>07:11:00</td>
<td>2384</td>
<td>1.1418</td>
<td>1.1293</td>
<td>.0125</td>
<td>.0115</td>
<td>29.8614</td>
<td>.0125</td>
<td>.0115</td>
<td>.0000</td>
<td>0</td>
<td>.0000</td>
<td>.0434</td>
<td></td>
<td></td>
</tr>
<tr>
<td>07:11:30</td>
<td>1945</td>
<td>3.4445</td>
<td>3.4287</td>
<td>.0158</td>
<td>.0117</td>
<td>30.8260</td>
<td>.0158</td>
<td>.0117</td>
<td>.0000</td>
<td>0</td>
<td>.0000</td>
<td>.3032</td>
<td></td>
<td></td>
</tr>
<tr>
<td>07:12:00</td>
<td>2446</td>
<td>2.4340</td>
<td>2.4223</td>
<td>.0117</td>
<td>.0106</td>
<td>28.6731</td>
<td>.0117</td>
<td>.0106</td>
<td>.0000</td>
<td>0</td>
<td>.0000</td>
<td>.1436</td>
<td></td>
<td></td>
</tr>
<tr>
<td>07:12:30</td>
<td>3240</td>
<td>1.7993</td>
<td>1.7902</td>
<td>.0091</td>
<td>.0086</td>
<td>29.5015</td>
<td>.0091</td>
<td>.0086</td>
<td>.0000</td>
<td>0</td>
<td>.0000</td>
<td>.0030</td>
<td></td>
<td></td>
</tr>
<tr>
<td>07:13:00</td>
<td>3051</td>
<td>.6163</td>
<td>.6072</td>
<td>.0091</td>
<td>.0086</td>
<td>27.8617</td>
<td>.0091</td>
<td>.0086</td>
<td>.0000</td>
<td>0</td>
<td>.0000</td>
<td>.0000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>07:13:30</td>
<td>3252</td>
<td>.0753</td>
<td>.0661</td>
<td>.0091</td>
<td>.0086</td>
<td>29.7519</td>
<td>.0091</td>
<td>.0086</td>
<td>.0000</td>
<td>0</td>
<td>.0000</td>
<td>.0000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>07:14:00</td>
<td>3258</td>
<td>.1391</td>
<td>.1300</td>
<td>.0091</td>
<td>.0086</td>
<td>29.6127</td>
<td>.0091</td>
<td>.0086</td>
<td>.0000</td>
<td>0</td>
<td>.0000</td>
<td>.0000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>07:14:30</td>
<td>3258</td>
<td>.1640</td>
<td>.1548</td>
<td>.0091</td>
<td>.0086</td>
<td>29.6975</td>
<td>.0091</td>
<td>.0086</td>
<td>.0000</td>
<td>0</td>
<td>.0000</td>
<td>.0000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Notice how the QR CPU time and the QR Disp time both suddenly increase. Given that the suspend time is almost all waiting for dispatch on the QR, it is clear that this sudden increase in QR Disp time has something to do with causing the MXT.
• Here we have the transactions summarized on 1-second intervals.
• With this we see that the point where the QR Disp and QR CPU times suddenly increased is actually at 07:09:58.
Prior to 07:09:58, there was a balance between transaction arrival rate and QR Disp time. Just enough transactions were arriving to keep the QR TCB totally busy. The 33% increase in QR Disp per task breaks that balance. Now the transactions are arriving faster than they can get their QR TCB time. So they back up.
RMFIII

- The problem is caused by transactions in IYNXK suddenly starting to use significantly more CPU at 7:09:58.
- Maybe RMFIII will yield some clues to help explain why that happened.
Type ‘2’ for Selection and press ENTER.
IBM Software Group

RMF Job Report Selection Menu

Selection ==> 5

Enter selection number or command and jobname for desired job report.

Jobname ==> IYNXK

1 DEVJ  Delay caused by devices  (DVJ)
1A DSNJ  .  Data set level  (DSJ)
2 ENQJ  Delay caused by ENQ  (EJ)
3 HSMJ  Delay caused by HSM  (HJ)
4 JESJ  Delay caused by JES  (JJ)
5 JOB  Delay caused by primary reason  (DELAYJ)
6 MNTJ  Delay caused by volume mount  (MTJ)
7 MSGJ  Delay caused by operator reply  (MSJ)
8 PROCJ  Delay caused by processor  (PJ)
9 QSCJ  Delay caused by QUIESCE via RESET command  (QJ)
10 STORJ  Delay caused by storage  (SJ)
11 XCFJ  Delay caused by XCF  (XJ)

These reports can also be selected by placing the cursor on the corresponding delay reason column of the DELAY or JOB reports and pressing ENTER or by using the commands from any panel.

F1=HELP  F2=SPLIT  F3=END  F4=RETURN  F5=RFIND  F6=TOGGLE
F7=UP  F8=DOWN  F9=SWAP  F10=BREF  F11=FREF  F12=RETRIEVE

• Type ‘5’ for Selection and ‘IYNXK’ for Jobname and press ENTER.
Note the Time towards the upper right corner. You can use F10 and F11 to scroll backwards and forwards through time.

Note the Range. That is the number of seconds in the interval.

On this page, the information covers from 07.08.20 to 07.10.00.
- You can also overtype Time to get to the time you want.
- When you do that, keep an eye on Range. It might double. Overtype Range to get it back to the smaller Range.
07.08.20 is the 100 second interval before the MXT began. (MXT began right around 07.10.08. The suddenly higher CPU began at 07.09.58.)

IYNXK is using 91% Processor and is delayed 9% for processor. IYNXJ is using 9% Processor.
The 07.10.00 interval is mostly all in the MXT period. IYNXK hasn’t changed much.

But IYNXJ is using 91% processor. That is a lot more than the prior interval. Let’s have a look at CPU.
• Type ‘3’ for Selection and press ENTER.
| IBM Software Group |

**RMF Resource Report Selection Menu**

- **Selection ==> 1A**

Enter selection number or command for desired report.

- Processor 1 PROC Processor delays (PD)
- 1A PROCU Processor usage (PU)
- Device 2 DEV Device delays (DD)
- 3 DEVR Device resource (DR)
- 3A DSND .. Data set level by DSN (DSN)
- 3B DSNV .. Data set level by volume (DSV)
- Enqueue 4 ENQ Enqueue delays (ED)
- 5 ENQR Enqueue resource (ER)
- Storage 6 STOR Storage delays for each job (SD)
- 7 STORF Storage usage by frames (SF)
- 7A STORM Storage usage by memory objects (SM)
- 8 STORR Storage usage for each resource (SR)
- 9 STORS Storage summary for each group (SS)
- 10 STORC Common storage summary (SC)
- 11 STORCR Common storage remaining (SCR)
- I/O Subsystem 12 CHANNEL Channel path activity (CH)
- 13 IOQUEUE I/O queuing activity (IQ)

F1=HELP F2=SPLIT F3=END F4=RETURN F5=RFINDE F6=TOGGLE
F7=UP F8=DOWN F9=SWAP F10=BREF F11=FREF F12=RETRIEVE

- Type ‘1A’ for Selection and press ENTER.
With the Time set to 07.08.20, the interval before the MXT, we see that IYNXK was using most of a processor, and IYNXJ was using 3 percent of a processor.

Press F11 to go to the next interval.
• During the MXT interval, IYNXJ is also using most of a processor.
• Could that cause transactions in IYNXK to use more CPU?
Systrace Perfdata

- Systrace Perfdata is an IPCS command that gives similar information to RMFIII regarding how much CPU is being used and what jobs are using it.
- Systrace Perfdata is new and newly documented at z/OS 1.12.
- We’ll look at the SLIP dump triggered by the “Above_80_percent_of_MXT” message.
- We’ll look at the dump of IYNXK taken after the problem was over, while it was doing its normal workload.
- This is on the SLIP dump.
- ENTER systrace perfdata
Systrace Perfdata processes the system trace.

We see that there are 2 processors doing work in the system trace.

And each of those has trace covering about .9 seconds from 06:10:14.0 to 06:10:14.9.
Idle Time of 0.00000 means that both processors were totally busy during the .9 seconds of systrace. There was never a moment when either had nothing to do.

Use F8 to scroll down to see what jobs are using those 1.8 seconds of CPU time.
Here we see that IYNXJ and IYNXK are together using up most of the 1.8 seconds of CPU time. They are each using most of a processor.

Now let’s take a look at the normal dump.
Here we see that each processor covers about 1.7 seconds of time.

And we see there is significant Idle time, almost 1 processors worth of idle time.

Scroll down to the next page.
IYNXK is using about 1 processors worth of CPU. And that is about it.

So that squares with RMFIII. During the problem, IYNXJ and IYNXK are each using most of a processor. Before and after the problem, IYNXK is using about 1 processor and the other processor is pretty much idle.
And the answer is……

- It looks like the LPAR is about 50% busy when everything is fine. And it is 100% busy when the problem happens. Can that cause transactions to suddenly use 33% more CPU?
- Clues point us to IYNXJ. Let’s take a look at the SMF110 data there to see what suddenly started using CPU.
This is a slightly tweaked DISPSUM form summarizing on 1-second intervals in IYNXJ.

At exactly 07:09:58, SOAK transactions began.

They are using a total of about .9 seconds of CPU per second, almost a whole processor. So that is why IYNXJ suddenly started using about 1 processors worth of CPU.
• The SOAK transaction does a loop of about 15 EXEC CICS GETMAIN followed by EXEC CICS FREEMAIN to get and free 20 Meg of EDSA, and it specifies INITIMG.

• INITIMG causes CICS, on every getmain, to write to every page of that 20 Meg.

• Part of the reason IYNXK transactions suddenly use more CPU is because the LPAR suddenly goes from 50% busy to 100% busy. At 50% busy as compared to 100% busy, the high-speed cache is more likely to always contain the pages of storage the instructions need. That is even more true given the fact that the SOAK transactions in IYNXJ are constantly writing to 20 Meg of storage. The constantly touching of the 20 Meg is making it so that the IYNXK transactions are constantly finding that the storage they need is not in the high-speed cache. That slows the IYNXK transactions down.
So what did you get?

- A neat new tool to put out console messages to expose MXT and near MXT
- A way to get a dump on MXT or near MXT
- A CICS Dispatcher refresher
- A way to approach response time spikes using SMF110 data
- A taste of how to make use of RMFIII
- A new IPCS tool: systrace perfdata
- An interesting reason why average CPU per transaction may vary from moment to moment
Questions and Answers