

MQ Problem Determination with Tracing on Linux

Tim Zielke

Introduction and Agenda

N

My Background:

O

- I have been in IT for 17 years with Hewitt Associates/Aon
- First 13 years mainly on the Mainframe COBOL application side
- Last 4 years as a CICS/MQ systems programmer
- Last 8 years also working with Linux

T

Session Agenda:

E

Using the Linux x86 platform, we will cover the following topics that can help with MQ problem determination:

S

- MQ API Tracing (also the MH06 Trace Tools supportpac)
- Application Activity Trace
- Helpful Linux x86 internals and commands for MQ problem determination

MQ API Tracing on Linux - Overview

N

Overview:

O

MQ API tracing is a debugging tool that comes with WebSphere MQ. An MQ API trace of an MQ application will include all of the API calls (i.e. MQOPEN, MQPUT, etc.) that the application makes, including the input and return data for each API call. This API data is very helpful in MQ problem determination, as it allows you to see what input data your application is passing to MQ and what return data your application is getting back from MQ. The MQ API trace can be cryptic to read, but we will cover a trace tool (mqtrcfrmt) that can significantly aid in reading MQ API traces much more quickly and accurately. We will do this, using Linux x86 as our platform.

T

E

S

MQ API Tracing – Example with amqspout

N

- Turn on an API trace for the amqspout program

```
strmqtrc -m qmgr -t api -p amqspout
```

O

- Run the amqspout program on a TCZ.TEST1 queue, and do two PUTs to the queue, and then end the program.

- NOTE: By default, trace writes out on Linux x86 to a file like:

```
/var/mqm/trace/AMQ16884.0.TRC (where 16884 = pid)
```

T

- Turn off the tracing

```
endmqtrc -a
```

E

- Format the trace

```
dspmqtrc AMQ16884.0.TRC > AMQ16884.0.FMT
```

- See handout #1 for contents of AMQ16884.0.FMT

S

Orientation in Reading an MQ API Trace

N

O

T

E

S

- Lines 3 – 27 have the trace header information.
- Line 33 shows the following trace data will have a microsecond time stamp, process.thread, and then API trace data.
- Lines 48 – 90 are an example of an MQOPEN API call. The trace records immediately following the “MQOPEN >>” on line 48 are the input data before entering the MQOPEN API. The trace records immediately following the “MQOPEN <<” on line 69 are the output data after exiting the MQOPEN API. Note that some data (i.e. ObjDesc) is both input and output data. Options is just input data. Compcode is just output data.
- Note for the Objdesc (lines 51 - 62), this MQ API data structure is printed in the raw hex data format, with each 16 byte line formatted to ASCII directly to the right.
- The rest of the API trace contains the 2 MQPUTs, and MQCLOSE, MQDISC.

Endianness – Little Endian (x86)

N

Endianness is the byte ordering of a CPU for multi-byte binary data. For reading MQ traces, it is helpful to understand Little endianness and Big endianness.

O

Example: x'01400006' stored on a Little endian processor (x86)

T

A Little endian CPU (x86) will store this 4 byte value at the starting memory address (i.e. address x'0000A010') from the least significant byte to most significant byte, or little end first.

E

```
address 0000A010 = x'06'  
address 0000A011 = x'00'  
address 0000A012 = x'40'  
address 0000A013 = x'01'
```

S

When looking at an MQ trace, this would appear as 06004001. This looks intuitively “reversed” when reading the trace.

Endianness – Big Endian (i.e. SPARC)

N

Example: x'01400006' stored on a Big endian processor (SPARC)

O

A Big endian CPU (SPARC) will store this 4 byte value at the starting memory address (i.e. address x'0000A010') from the most significant byte to least significant byte, or big end first.

T

```
address 0000A010 = x'01'  
address 0000A011 = x'40'  
address 0000A012 = x'00'  
address 0000A013 = x'06'
```

E

When looking at an MQ trace, this would appear as 01400006. This looks intuitively “normal” when reading the trace.

S

MQ Tracing – Reading a Data Structure

N

O

T

E

S

- MQ data structures such as Objdesc, Msgdesc, Putmsgopts, etc. appear in the trace. The MQ data structures follow a format of a 4 byte character structure id, a 4 byte binary integer version id, and then subsequent fields. The layouts of the data structures can be found in the MQ manual.

					1	2	3
51	13:15:37.742885	16884.1	Objdesc:				
52	13:15:37.742887	16884.1	0x0000:	4f442020	01000000	01000000	54435a2e
53	13:15:37.742887	16884.1	0x0010:	54455354	31000000	00000000	00000000

Field 1 is Structure Id (MQCHAR4) = x'4f442020' = "OD "

Field 2 is Version (MQLONG) = x'01000000' = 1

Field 3 is Object Type (MQLONG) = x'01000000' = 1 (MQOT_Q or Queue Object Type)

Remember to reverse bytes for binary fields, since this trace is little endian (x86):

Version:

01 00 00 00



= Version is 1

MQ Tracing – Reading an Options Field

N

- Reading Open Options on line 64

```
63          13:15:37.742888      16884.1      Options:
64          13:15:37.742889      16884.1      0x0000: 10200000
```

O

Reverse bytes for binary integer fields, since this is little endian:

Options (MQLONG):

10 20 00 00



00 00 20 10 = Options is x'00002010' = 8208

T

To convert 8208 to its open options constant values, find the largest open option value that is closest to or equal to 8208 and subtract that value. Continue this process, until you reach 0.

8192 = MQOO_FAIL_IF QUIESCING

8208 - 8192 = 16

16 = MQOO_OUTPUT

16 - 16 = 0

Therefore, 8208 = MQOO_FAIL_IF QUIESCING, MQOO_OUTPUT

E

S

MQ Tracing – mqtrcfrmt tool in MH06

N

- mqtrcfrmt is a trace tool that comes with the MH06 supportpac. It will help you read a trace by expanding the MQ data structures by labeling the fields and include constant expansions. Executables are provided for Linux x86, Solaris Sparc, and Windows.

O

- Using the mqtrcfrmt tool:

```
mqtrcfrmt.linux AMQ16884.0.FMT AMQ16884.0.FMT2
```

T

- See handout #2 for contents of AMQ16884.0.FMT2
- User customizable API summary trace from AMQ16884.0.FMT2

E

```
egrep '( >>$| <<$|Hconn=|Hobj=|Compcode=|Reason=|Hmsg=|Actual Name=|  
Value=|Options=|Type=|ObjectName |ResolvedQName |Persistence ) '  
AMQ16884.0.FMT2
```

S

- See handout #3 for results of this API summary trace

MQ Tracing - AMQ16884.0.FMT2

N

O

T

E

S

```
59      13:15:37.742885      16884.1      Objdesc:
60      13:15:37.742887      16884.1      0x0000:  4f442020 01000000 01000000 54435a2e
61      13:15:37.742887      16884.1      0x0010:  54455354 31000000 00000000 00000000
62      13:15:37.742887      16884.1      0x0020:  00000000 00000000 00000000 00000000
63      13:15:37.742887      16884.1      0x0030:  00000000 00000000 00000000 00000000
64      13:15:37.742887      16884.1      0x0040:  00000000 00000000 00000000 00000000
65      13:15:37.742887      16884.1      0x0050:  00000000 00000000 00000000 00000000
66      13:15:37.742887      16884.1      0x0060:  00000000 00000000 00000000 414d512e
67      13:15:37.742887      16884.1      0x0070:  2a000000 00000000 00000000 00000000
68      13:15:37.742887      16884.1      0x0080:  00000000 00000000 00000000 00000000
69      13:15:37.742887      16884.1      0x0090:  00000000 00000000 00000000 00000000
70      13:15:37.742887      16884.1      0x00a0:  00000000 00000000
71                                     Objdesc expanded (all fields):
72                                     StrucId (CHAR4)           : 'OD  '
73                                     x'4f442020'
74                                     Version (MQLONG)         : 1
75                                     x'01000000'
76                                     ObjectType (MQLONG)        : 1
77                                     x'01000000'
78                                     ObjectType MQOT_Q
79                                     ObjectName (MQCHAR48)       : 'TCZ.TEST1'
80                                     x'54435a2e5445535431
81                                     ObjectQMgrName (MQCHAR48)  : '.....'
82                                     x'00000000000000000000
83                                     DynamicQName (MQCHAR48)    : 'AMQ.*'
84                                     x'414d512e2a
85                                     AlternateUserId (MQCHAR12) : '.....'
86                                     x'00000000000000000000
```

MQ Tracing – API Summary Trace

N

```
mqm@MYSERVER123$ egrep '( >>$| <<$|Hconn=|Hobj=|Compcode=|Reason=|Hmsg=|Actual Name=|Value=|Options=|
Type=|ObjectName |ResolvedQName |Persistence )' AMQ16884.0.FMT2
```

O

```
13:15:37.742829 16884.1      CONN:1400006      MQCONN <<
                  16884.1      Hconn=06004001
                  16884.1      MQCNO.Options= (MQLONG)      : 256
                  16884.1      Options=MQCNO_SHARED_BINDING
                  16884.1      Compcode=0
                  16884.1      Reason=0
13:15:37.742881 16884.1      CONN:1400006      MQOPEN >>
                  16884.1      Hconn=06004001
                  16884.1      ObjectName (MQCHAR48)      :
'TCZ.TEST1.....'
                  16884.1      MQOO.Options= (MQLONG)      : 8208
                  16884.1      Options=MQOO_OUTPUT
                  16884.1      Options=MQOO_FAIL_IF QUIESCING
13:15:37.743138 16884.1      CONN:1400006      MQOPEN <<
                  16884.1      ObjectName (MQCHAR48)      :
'TCZ.TEST1.....'
                  16884.1      Hobj=02000000
                  16884.1      Compcode=0
                  16884.1      Reason=0
13:15:37.743176 16884.1      CONN:1400006      MQI:MQOPEN HConn=01400006 HObj=00000002 rc=00000000
ObjType=00000001 ObjName=TCZ.TEST1
```

T

E

S

MQ Tracing – Other Uses

N
O
T
E
S

- 1) General performance of API calls
 - API tracing provides microsecond timings in the trace record. By finding the API begin (i.e. MQGET >>) and the API end (i.e. reason field of MQGET <<) you can roughly calculate the time it took for the API MQGET to complete. Do note that tracing does add overhead to the timings.

```
48  13:15:37.742881    16884.1      CONN:1400006      MQOPEN >>
69  13:15:37.743138    16884.1      CONN:1400006      MQOPEN <<
88  13:15:37.743157    16884.1      CONN:1400006      Reason:
89  13:15:37.743158    16884.1      CONN:1400006      0x0000: 00000000
```

13:15:37.743158 - 13:15:37.742881 = 0.000277 seconds to complete for the MQOPEN

- mqapitrcstats tool in the MH06 Trace Tools supportpac will read an entire API trace and create a summary report of the response times of the open, close, get, put, and put1 API calls. Executables are provided for Linux x86, Solaris Sparc, and Windows.

MQ Tracing – Other Uses

N

2) Investigation of triggering issues

```
strmqtrc -m qmgr -t all -p runmqtrm
```

O

- The runmqtrm trace will record if/when the trigger message was read from the INITQ, if/when it started the application of your process, and the operating system return code from the start call. Also, this does not require that runmqtrm be run in the foreground.

T

E

S

Some Final MQ Tracing Notes

N

- Client applications can be traced, as well. You can either run a client trace on the client server (unfortunately, Java clients do not support this type of tracing) or trace the queue manager process that the SVRCONN channel is running on.

O

- Examples of client traces

T

1. `strmqtrc -t api -p prog1` (from client server)
2. `strmqtrc -m qmgr -t api -p amqrmppa` (from queue manager server)

E

S

Some Final MQ Tracing Notes – cont

N
O
T
E
S

- Tracing adds performance overhead and can create large files. Be judicious on the length of time that you run the trace and try and be selective with the options (i.e. `-t api -p prog1`) to reduce any unneeded output. Also, keep an eye on the size of your trace files and your space available on your trace file system (i.e. `/var/mqm`). You can also use the `strmqtrc -l (MaxSize in MB)` option to limit the size of your trace files, but this means that trace data can be overwritten and lost. The `-l` option keeps a current `AMQpppppp.qq.TRC` and a previous `AMQpppppp.qq.TRS` file.

```
strmqtrc -m qmgr -t api -p amqspout -l 1
```

- APAR IT01972 – Queue Manager trace is inadvertently turned off for an application thread with multiple shared connections after an `MQDISC` is called. End result is the potential for trace data loss. Targeted delivery of PTF is 7.1.0.6, 7.5.0.5, 8.0.0.1.

Application Activity Trace (AAT)

N

- The Application Activity Trace (AAT) was first introduced in 7.1. It provides detailed information of the behavior of applications connected to a queue manager, including their MQI call details.

O

- “Increasing the visibility of messages using WebSphere MQ Application Activity Trace” by Emma Bushby is an IBM DeveloperWorks article that does a good job in explaining the Application Activity Trace in detail.

T

- The AAT is another tool that can be helpful in MQ problem determination or application review, by giving you visibility to the inputs and outputs of your application API calls.

E

S

AAT – Usage Notes

N

- Applications write AAT records to the `SYSTEM.ADMIN.TRACE.ACTIVITY.QUEUE`.

O

- There is a hierarchy to turning ON/OFF the AAT:

1. `ACTVTRC` queue manager attribute (ON/OFF)
(overridden by)
2. `MQCNO_ACTIVITY_TRACE` connection options specified in an `MQCONN`
(NOTE: `ACTVCONO` queue manager attribute must be `ENABLED` for this to be checked, and the default value is `DISABLED`)
(overridden by)
3. Settings in a matching stanza in `mqat.ini` (located in `qm.ini` directory)

T

E

- In order to pick up a `mqat.ini` change dynamically in a running program, you need to toggle the `ACTVTRC` queue manager attribute (i.e. ON/OFF).

S

AAT – Viewing the Data

N

- MS0P supportpac (WebSphere MQ Explorer Extended Management Plugins) has an Application Activity Trace viewer.

O

- amqsact is a command line tool (sample code also provided) that can read the messages from the `SYSTEM.ADMIN.TRACE.ACTIVITY.QUEUE` and format them into summary and verbose reports.

T

- amqsactz on Capitalware's Sample WebSphere MQ C Code web site is a program that takes the amqsact sample code and provides the following enhancements:

E

1. Includes more data (i.e. Conn, Channel, etc.) on API one line summaries.
2. Includes `-r` option for helpful summary reports
3. Corrects a print formatting issue where a byte like `x'DF'` was printed as `x'FFFFFFDF'`.

S

AAT – amqsputc example

N

1) Add an ApplicationTrace stanza for amqsputc to the mqat.ini to turn on AAT tracing.

O

```
ApplicationTrace:      # Application specific settings stanza
  ApplClass=ALL         # Application type
                        #   Values:  (USER | MCA | ALL)
                        #   Default: USER
  ApplName=amqsputc     # Application name (may be wildcarded)
                        #   (matched to app name without path)
                        #   Default: *
  Trace=ON              # Activity trace switch for application
                        #   Values:  ( ON | OFF )
                        #   Default: OFF
  ActivityInterval=0    # Time interval between trace messages
                        #   Values:  0-999999999 (0=off)
                        #   Default: 0
  ActivityCount=0       # Number of operations between trace msgs
                        #   Values:  0-999999999 (0=off)
                        #   Default: 0
  TraceLevel=MEDIUM    # Amount of data traced for each operation
                        #   Values:  LOW | MEDIUM | HIGH
                        #   Default: MEDIUM
  TraceMessageData=0    # Amount of message data traced
                        #   Values:  0-104857600
                        #   Default: 0
```

T

E

S

AAT – amqsputc example

N 2) Run the amqsputc sample program.

```
mqm$ export MQSERVER='CLIENT.TO.SERVER/TCP/SERVER01'
mqm$ amqsputc TCZ.TEST1
Sample AMQSPUT0 start
target queue is TCZ.TEST1
test1
test2
test3
test4
test5

Sample AMQSPUT0 end
```

O
T
E 3) Update ApplicationTrace stanza for amqsputc in the mqat.ini to turn off AAT tracing.

S NOTE: If instead, amqsputc was to continue to run and you turned the trace off with the mqat.ini change, you would need to toggle the ACTVTRC queue manager attribute ON/OFF to have amqsputc pick up the mqat.ini change.

AAT – amqsactz reports

N

4) Use amqsactz to view AAT data, by generating 3 reports:

O

1.amqsactz.out – non-verbose report with –r summary information

2.amqsactz_1LS.out – API one line summaries selected from amqsactz.out

3.amqsactv_v.out – verbose report

T

■ amqsactz.out – non-verbose report, includes –r summary output at bottom of report

```
amqsactz -r -b > amqsactz.out
```

E

S

AAT - amqsactz.out

N

MonitoringType: MQI Activity Trace RecordNum: 0 ← 1

Correl_id:

00000000: 414D 5143 5345 5256 5245 3031 2E4D 5154 'AMQCSERVER01.MQT'

00000010: 53E3 C453 010A F420 'S..S...'

QueueManager: 'SERVER01.MQTEST1'

Host Name: 'server01'

IntervalStartDate: '2014-08-28'

IntervalStartTime: '09:24:29'

IntervalEndDate: '2014-08-28'

IntervalEndTime: '09:24:35'

CommandLevel: 750

SeqNumber: 0

ApplicationName: 'amqsputc' ← 2

Application Type: MQAT_UNIX

ApplicationPid: 24912 ← 3

UserId: 'mqm'

API Caller Type: MQXACT_EXTERNAL

API Environment: MQXE_MCA_SVRCONN

Channel Name: 'CLIENT.TO.SERVER' ← 4

ConnName: '127.0.0.1'

Channel Type: MQCHT_SVRCONN

Application Function: ''

Appl Function Type: MQFUN_TYPE_UNKNOWN

Trace Detail Level: 2

Trace Data Length: 0

Pointer size: 8

Platform: MQPL_UNIX

O

T

E

S

AAT - amqsactz.out

N

O

T

E

S

UserId: 'mqm'
API Caller Type: MQXACT_EXTERNAL
API Environment: MQXE_MCA_SVRCONN
Channel Name: 'CLIENT.TO.SERVER'
ConnName: '127.0.0.1'
Channel Type: MQCHT_SVRCONN
Application Function: ''
Appl Function Type: MQFUN_TYPE_UNKNOWN
Trace Detail Level: 2
Trace Data Length: 0
Pointer size: 8
Platform: MQPL_UNIX

```
=====
EYEC RecordNum Pid  Tid  Conn          Channel Name      Date       Time       Operation  MQRC HObj
(ObjName)
1LS= 4 24912 2606 53E3C453010AF420 CLIENT.TO.SERVER 2014-08-28 09:24:29 MQXF_CONNX 0000 -
1LS= 0 24912 2606 53E3C453010AF420 CLIENT.TO.SERVER 2014-08-28 09:24:29 MQXF_OPEN 0000 2
(TCZ.TEST1)
1LS= 0 24912 2606 53E3C453010AF420 CLIENT.TO.SERVER 2014-08-28 09:24:31 MQXF_PUT 0000 2
(TCZ.TEST1)
1LS= 0 24912 2606 53E3C453010AF420 CLIENT.TO.SERVER 2014-08-28 09:24:31 MQXF_PUT 0000 2
(TCZ.TEST1)
1LS= 0 24912 2606 53E3C453010AF420 CLIENT.TO.SERVER 2014-08-28 09:24:32 MQXF_PUT 0000 2
(TCZ.TEST1)
1LS= 0 24912 2606 53E3C453010AF420 CLIENT.TO.SERVER 2014-08-28 09:24:33 MQXF_PUT 0000 2
(TCZ.TEST1)
1LS= 0 24912 2606 53E3C453010AF420 CLIENT.TO.SERVER 2014-08-28 09:24:35 MQXF_PUT 0000 2
(TCZ.TEST1)
1LS= 0 24912 2606 53E3C453010AF420 CLIENT.TO.SERVER 2014-08-28 09:24:35 MQXF_CLOSE 0000 2
(TCZ.TEST1)
1LS= 0 24912 2606 53E3C453010AF420 CLIENT.TO.SERVER 2014-08-28 09:24:35 MQXF_DISC 0000 -
=====
```


AAT - amqsactz.out (-r summary option)

N

Application Summary Report

Pid	ApplicationName	UserId	Tid Count	MQI Count
24912	amqsputc	mqm	1	11

O

Application Objects Referenced Report

pid: 24912 ApplicationName: amqsputc UserId: mqm referenced the following objects:
ObjName: TCZ.TEST1 Count: 7

T

Application Objects Options Report

Options tracked are conn, open, get, put, close, callback, sub, subrq

E

pid: 24912 ApplicationName: amqsputc UserId: mqm referenced the following options by object:

Object Name: TCZ.TEST1

Open Options: 8208 Count: 1

MQOO_OUTPUT

MQOO_FAIL_IF QUIESCING

Put Options: 8260 Count: 5

MQPMO_NO_SYNCPOINT

MQPMO_NEW_MSG_ID

MQPMO_FAIL_IF QUIESCING

Close Options: 0 Count: 1

MQCO_NONE

MQCO_IMMEDIATE

S

AAT - amqsactz.out (-r summary option)

N

```
=====
Application Channels Referenced Report
=====
```

```
pid: 24912    ApplicationName: amqsputc    UserId: mqm    referenced the following channels:
  ChannelName: CLIENT.TO.SERVER    Count: 11
```

O

```
=====
Application Operations Executed Report
=====
```

```
pid: 24912    ApplicationName: amqsputc    UserId: mqm    executed the
following operations:
  Operation: MQXF_BACK    Count: 1
  Operation: MQXF_CLOSE    Count: 1
  Operation: MQXF_CONNX    Count: 1
  Operation: MQXF_DISC    Count: 2
  Operation: MQXF_OPEN    Count: 1
  Operation: MQXF_PUT    Count: 5
```

T

E

S

AAT - amqsactz.out (-r summary option)

N

O

T

E

S

```
=====
Application Operations Options Report
```

```
Options tracked are conn, open, get, put, close, callback, sub, subrq
=====
```

```
pid: 24912   ApplicationName: amqsputc   UserId: mqm   referenced the following options by operations:
```

```
Operation: MQXF_CLOSE
```

```
Close Options: 0           Count: 1
```

```
MQCO_NONE
```

```
MQCO_IMMEDIATE
```

```
Operation: MQXF_CONNX
```

```
Connect Options: 320       Count: 1
```

```
MQCNO_HANDLE_SHARE_BLOCK
```

```
MQCNO_SHARED_BINDING
```

```
Operation: MQXF_OPEN
```

```
Open Options: 8208        Count: 1
```

```
MQOO_OUTPUT
```

```
MQOO_FAIL_IF QUIESCING
```

```
Operation: MQXF_PUT
```

```
Put Options: 8260         Count: 5
```

```
MQPMO_NO_SYNCPOINT
```

```
MQPMO_NEW_MSG_ID
```

```
MQPMO_FAIL_IF QUIESCING
```

AAT – amqsactz reports

N
O
T
E
S

- Remember, each API summary line had a 1LS= eye catcher text in it.

```
=====
EYEC RecordNum Pid  Tid  Conn                Channel Name      Date       Time       Operation  MQRC HObj
  (ObjName)
1LS= 0                24912 2606 53E3C453010AF420 CLIENT.TO.SERVER 2014-08-28 09:24:29 MQXF_CONNX 0000  -
1LS= 0                24912 2606 53E3C453010AF420 CLIENT.TO.SERVER 2014-08-28 09:24:29 MQXF_OPEN  0000  2
  (TCZ.TEST1)
```

- amqsactz_1LS.out - Use the 1LS= to grep out all the API one line summary records into a report.

```
grep 1LS= amqsactz.out > amqsactz_1LS.out
```

AAT - amqsactz_1LS.out

N
O
T
E
S

	1 (RecordNum)	2 (Pid)	3 (Tid)	4 (Conn)	5 (Channel)				
1LS= 0	24912	2606	53E3C453010AF420	CLIENT.TO.SERVER	2014-08-28 09:24:29	MQXF_CONNX	0000	-	
1LS= 0	24912	2606	53E3C453010AF420	CLIENT.TO.SERVER	2014-08-28 09:24:29	MQXF_OPEN	0000	2	
(TCZ.TEST1)									
1LS= 0	24912	2606	53E3C453010AF420	CLIENT.TO.SERVER	2014-08-28 09:24:31	MQXF_PUT	0000	2	
(TCZ.TEST1)									
1LS= 0	24912	2606	53E3C453010AF420	CLIENT.TO.SERVER	2014-08-28 09:24:31	MQXF_PUT	0000	2	
(TCZ.TEST1)									
1LS= 0	24912	2606	53E3C453010AF420	CLIENT.TO.SERVER	2014-08-28 09:24:32	MQXF_PUT	0000	2	
(TCZ.TEST1)									
1LS= 0	24912	2606	53E3C453010AF420	CLIENT.TO.SERVER	2014-08-28 09:24:33	MQXF_PUT	0000	2	
(TCZ.TEST1)									
1LS= 0	24912	2606	53E3C453010AF420	CLIENT.TO.SERVER	2014-08-28 09:24:35	MQXF_PUT	0000	2	
(TCZ.TEST1)									
1LS= 0	24912	2606	53E3C453010AF420	CLIENT.TO.SERVER	2014-08-28 09:24:35	MQXF_CLOSE	0000	2	
(TCZ.TEST1)									
1LS= 0	24912	2606	53E3C453010AF420	CLIENT.TO.SERVER	2014-08-28 09:24:35	MQXF_DISC	0000	-	
1LS= 1	24912	2606	53E3C453010AF420	CLIENT.TO.SERVER	2014-08-28 09:24:35	MQXF_BACK	0000	-	
1LS= 1	24912	2606	53E3C453010AF420	CLIENT.TO.SERVER	2014-08-28 09:24:35	MQXF_DISC	0000	-	

6 (ObjName) NOTE: ObjName is being line wrapped, and is on same line as MQXF_CLOSE)

AAT – amqsactz reports

N

- amqsactz_v.out – verbose report for each AAT record

```
amqsactz -v -b > amqsactz_v.out
```

O

T

E

S

AAT - amqsactz_v.out

N

O

T

E

S

MonitoringType: MQI Activity Trace RecordNum: 0

MQI Operation: 2

Operation Id: MQXF_PUT

ApplicationTid: 2606

OperationDate: '2014-08-28'

OperationTime: '09:24:31'

High Res Time: 1409235871018640

Completion Code: MQCC_OK

Reason Code: 0

Hobj: 2

Put Options: 8260 ← 1

Msg length: 5

Known_dest_count: 1

Unknown_dest_count: 0

Invalid_dest_count: 0

Object_type: MQOT_Q

Object_name: 'TCZ.TEST1' ← 2

Object_Q_mgr_name: ''

Resolved_Q_Name: 'TCZ.TEST1'

Resolved_Q_mgr: 'SERVER01.MQTEST1'

Resolved_local_Q_name: 'TCZ.TEST1'

Resolved_local_Q_mgr: 'SERVER01.MQTEST1'

Resolved_type: MQOT_Q

Report Options: 0

Msg_type: MQMT_DATAGRAM

Expiry: -1 ← 3

Format_name: 'MQSTR'

Priority: -1 ← 4

Persistence: 2

Msg_id:

00000000: 414D 5120 5345 5256 5245 3031 2E4D 5154 'AMQ SERVER01.MQT'

00000010: 53E3 C453 020A F420 'S..S...' ' '

Linux Internals - /proc file system

N

- The /proc file system is a virtual file system that allows you access to internal kernel data.

O

- /proc/pid/envIRON will show you the environment variables for a pid when the process was created. NOTE: Later changes to the environment after the process is created are not reflected here. See handout #8.

T

- /proc/pid/limits will show the user limits for a process. Later versions of the Linux kernel (I believe 2.6.32) allow you to dynamically update the limits for a running process. For example, you can increase the Max open files setting for a running queue manager process. See handout #9.

E

- /proc/pid/fd will show the file descriptors for the process. You could see which files that the process has open, or how many files are open. See handout #10.

S

Linux Commands - strace

N

- strace will trace the system calls (i.e. open, write, etc.) that a process is making. This can be helpful to see internally what a process is doing for problem determination. You need the proper security to strace a process. For processes running under the mqm id, you need to be the mqm id.

O

- Possible uses are:
 1. strace the start up of the queue manager to see if an exit is being invoked.
 2. strace an actively running process that seems to be hung.
 3. strace a problematic MQ command.

T

- Helpful pieces to search for in the strace output:
 1. file names
 2. programs being executed
 3. error messages being written

E

S

See handout #11.

Questions & Answers



MQ Problem Determination with Tracing on Linux

Tim Zielke

Capitalware's MQ Technical Conference v2.0.1.4

Introduction and Agenda

N

My Background:

O

- I have been in IT for 17 years with Hewitt Associates/Aon
- First 13 years mainly on the Mainframe COBOL application side
- Last 4 years as a CICS/MQ systems programmer
- Last 8 years also working with Linux

T

Session Agenda:

E

Using the Linux x86 platform, we will cover the following topics that can help with MQ problem determination:

S

- MQ API Tracing (also the MH06 Trace Tools supportpac)
- Application Activity Trace
- Helpful Linux x86 internals and commands for MQ problem determination

MQ API Tracing on Linux - Overview

N Overview:

O MQ API tracing is a debugging tool that comes with WebSphere MQ. An MQ API trace of an MQ application will include all of the API calls (i.e. MQOPEN, MQPUT, etc.) that the application makes, including the input and return data for each API call. This API data is very helpful in MQ problem determination, as it allows you to see what input data your application is passing to MQ and what return data your application is getting back from MQ. The MQ API trace can be cryptic to read, but we will cover a trace tool (mqtrcfrmt) that can significantly aid in reading MQ API traces much more quickly and accurately. We will do this, using Linux x86 as our platform.

T

E

S

MQ API Tracing – Example with amqspout

N

- Turn on an API trace for the amqspout program

```
strmqtrc -m qmgr -t api -p amqspout
```

O

- Run the amqspout program on a TCZ.TEST1 queue, and do two PUTs to the queue, and then end the program.

T

- NOTE: By default, trace writes out on Linux x86 to a file like:

```
/var/mqm/trace/AMQ16884.0.TRC (where 16884 = pid)
```

E

- Turn off the tracing

```
endmqtrc -a
```

S

- Format the trace

```
dspmqtrc AMQ16884.0.TRC > AMQ16884.0.FMT
```

- See handout #1 for contents of AMQ16884.0.FMT

Orientation in Reading an MQ API Trace

- | | |
|---|--|
| N | ▪ Lines 3 – 27 have the trace header information. |
| O | ▪ Line 33 shows the following trace data will have a microsecond time stamp, process.thread, and then API trace data. |
| T | ▪ Lines 48 – 90 are an example of an MQOPEN API call. The trace records immediately following the “MQOPEN >>” on line 48 are the input data before entering the MQOPEN API. The trace records immediately following the “MQOPEN <<” on line 69 are the output data after exiting the MQOPEN API. Note that some data (i.e. ObjDesc) is both input and output data. Options is just input data. Compcode is just output data. |
| E | ▪ Note for the Objdesc (lines 51 - 62), this MQ API data structure is printed in the raw hex data format, with each 16 byte line formatted to ASCII directly to the right. |
| S | ▪ The rest of the API trace contains the 2 MQPUTs, and MQCLOSE, MQDISC. |

Endianness – Little Endian (x86)

- N** Endianness is the byte ordering of a CPU for multi-byte binary data. For reading MQ traces, it is helpful to understand Little endianness and Big endianness.
- O** Example: x'01400006' stored on a Little endian processor (x86)
- T** A Little endian CPU (x86) will store this 4 byte value at the starting memory address (i.e. address x'0000A010') from the least significant byte to most significant byte, or little end first.
- E**
- S**
- ```
address 0000A010 = x'06'
address 0000A011 = x'00'
address 0000A012 = x'40'
address 0000A013 = x'01'
```
- When looking at an MQ trace, this would appear as 06004001. This looks intuitively “reversed” when reading the trace.



## Endianness – Big Endian (i.e. SPARC)

**N** Example: x'01400006' stored on a Big endian processor (SPARC)

**O** A Big endian CPU (SPARC) will store this 4 byte value at the starting memory address (i.e. address x'0000A010') from the most significant byte to least significant byte, or big end first.

**T**  
`address 0000A010 = x'01'`  
`address 0000A011 = x'40'`  
`address 0000A012 = x'00'`  
`address 0000A013 = x'06'`

**E** When looking at an MQ trace, this would appear as 01400006. This looks intuitively “normal” when reading the trace.

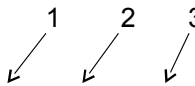
**S**

## MQ Tracing – Reading a Data Structure

- N** ■ MQ data structures such as Objdesc, Msgdesc, Putmsgopts, etc. appear in the trace. The MQ data structures follow a format of a 4 byte character structure id, a 4 byte binary integer version id, and then subsequent fields. The layouts of the data structures can be found in the MQ manual.
- O**

**T**

|    |                 |         |          |          |          |          |          |
|----|-----------------|---------|----------|----------|----------|----------|----------|
| 51 | 13:15:37.742885 | 16884.1 | Objdesc: |          |          |          |          |
| 52 | 13:15:37.742887 | 16884.1 | 0x0000:  | 4f442020 | 01000000 | 01000000 | 54435a2e |
| 53 | 13:15:37.742887 | 16884.1 | 0x0010:  | 54455354 | 31000000 | 00000000 | 00000000 |




**E**

Field 1 is Structure Id (MQCHAR4) = x'4f442020' = "OD "  
Field 2 is Version (MQLONG) = x'01000000' = 1  
Field 3 is Object Type (MQLONG) = x'01000000' = 1 (MQOT\_Q or Queue Object Type)

Remember to reverse bytes for binary fields, since this trace is little endian (x86):

**S**

Version:  
01 00 00 00  
00 00 00 01 = Version is 1



# MQ Tracing – Reading an Options Field

N

## ■ Reading Open Options on line 64

```
63 13:15:37.742888 16884.1 Options:
64 13:15:37.742889 16884.1 0x0000: 10200000
```

O

Reverse bytes for binary integer fields, since this is little endian:

Options (MQLONG):

10 20 00 00

← →

00 00 20 10 = Options is x'00002010' = 8208

T

To convert 8208 to its open options constant values, find the largest open option value that is closest to or equal to 8208 and subtract that value. Continue this process, until you reach 0.

E

8192 = MQOO\_FAIL\_IF QUIESCING

8208 - 8192 = 16

16 = MQOO\_OUTPUT

16 - 16 = 0

S

Therefore, 8208 = MQOO\_FAIL\_IF QUIESCING, MQOO\_OUTPUT

## MQ Tracing – mqtrcfrmt tool in MH06

N  
O  
T  
E  
S

- mqtrcfrmt is a trace tool that comes with the MH06 supportpac. It will help you read a trace by expanding the MQ data structures by labeling the fields and include constant expansions. Executables are provided for Linux x86, Solaris Sparc, and Windows.

- Using the mqtrcfrmt tool:

```
mqtrcfrmt.linux AMQ16884.0.FMT AMQ16884.0.FMT2
```

- See handout #2 for contents of AMQ16884.0.FMT2
- User customizable API summary trace from AMQ16884.0.FMT2

```
egrep '(>>$| <<$|Hconn=|Hobj=|Compcode=|Reason=|Hmsg=|Actual Name=|
Value=|Options=|Type=|ObjectName |ResolvedQName |Persistence) '
AMQ16884.0.FMT2
```

- See handout #3 for results of this API summary trace

# MQ Tracing - AMQ16884.0.FMT2

N  
O  
T  
E  
S

|    |                 |         |                                             |
|----|-----------------|---------|---------------------------------------------|
| 59 | 13:15:37.742885 | 16884.1 | Objdesc:                                    |
| 60 | 13:15:37.742887 | 16884.1 | 0x0000: 4f442020 01000000 01000000 54435a2e |
| 61 | 13:15:37.742887 | 16884.1 | 0x0010: 54455354 31000000 00000000 00000000 |
| 62 | 13:15:37.742887 | 16884.1 | 0x0020: 00000000 00000000 00000000 00000000 |
| 63 | 13:15:37.742887 | 16884.1 | 0x0030: 00000000 00000000 00000000 00000000 |
| 64 | 13:15:37.742887 | 16884.1 | 0x0040: 00000000 00000000 00000000 00000000 |
| 65 | 13:15:37.742887 | 16884.1 | 0x0050: 00000000 00000000 00000000 00000000 |
| 66 | 13:15:37.742887 | 16884.1 | 0x0060: 00000000 00000000 00000000 414d512e |
| 67 | 13:15:37.742887 | 16884.1 | 0x0070: 2a000000 00000000 00000000 00000000 |
| 68 | 13:15:37.742887 | 16884.1 | 0x0080: 00000000 00000000 00000000 00000000 |
| 69 | 13:15:37.742887 | 16884.1 | 0x0090: 00000000 00000000 00000000 00000000 |
| 70 | 13:15:37.742887 | 16884.1 | 0x00a0: 00000000 00000000                   |
| 71 |                 | 16884.1 | Objdesc expanded (all fields):              |
| 72 |                 | 16884.1 | StrucId (CHAR4) : 'OD '                     |
| 73 |                 | 16884.1 | : x'4f442020'                               |
| 74 |                 | 16884.1 | Version (MQLONG) : 1                        |
| 75 |                 | 16884.1 | : x'01000000'                               |
| 76 |                 | 16884.1 | ObjectType (MQLONG) : 1                     |
| 77 |                 | 16884.1 | : x'01000000'                               |
| 78 |                 | 16884.1 | ObjectType MQOT_Q                           |
| 79 |                 | 16884.1 | ObjectName (MQCHAR48) : 'TCZ.TEST1'         |
| 80 |                 | 16884.1 | : x'54435a2e5445535431                      |
| 81 |                 | 16884.1 | ObjectQMgrName (MQCHAR48) : '.....'         |
| 82 |                 | 16884.1 | : x'00000000000000000000                    |
| 83 |                 | 16884.1 | DynamicQName (MQCHAR48) : 'AMQ.*            |
| 84 |                 | 16884.1 | : x'414d512e2a                              |
| 85 |                 | 16884.1 | AlternateUserId (MQCHAR12) : '.....'        |
| 86 |                 | 16884.1 | : x'00000000000000000000                    |

# MQ Tracing – API Summary Trace

N  
O  
T  
E  
S

```
mqm@MYSERVER123$ egrep '(>>$| <<$|Hconn=|Hobj=|Compcode=|Reason=|Hmsg=|Actual Name=|Value=|Options=|
Type=|ObjectName |ResolvedQName |Persistence)' AMQ16884.0.FMT2

13:15:37.742829 16884.1 CONN:1400006 MQCONN <<
16884.1 Hconn=06004001
16884.1 MQCNO.Options= (MQLONG) : 256
16884.1 Options=MQCNO_SHARED_BINDING
16884.1 Compcode=0
16884.1 Reason=0
13:15:37.742881 16884.1 CONN:1400006 MQOPEN >>
16884.1 Hconn=06004001
16884.1 ObjectName (MQCHAR48) :
'TCZ.TEST1.....'
16884.1 MQOO.Options= (MQLONG) : 8208
16884.1 Options=MQOO_OUTPUT
16884.1 Options=MQOO_FAIL_IF QUIESCING
13:15:37.743138 16884.1 CONN:1400006 MQOPEN <<
16884.1 ObjectName (MQCHAR48) :
'TCZ.TEST1.....'
16884.1 Hobj=02000000
16884.1 Compcode=0
16884.1 Reason=0
13:15:37.743176 16884.1 CONN:1400006 MQI:MQOPEN HConn=01400006 HObj=00000002 rc=00000000
ObjType=00000001 ObjName=TCZ.TEST1
```

# MQ Tracing – Other Uses

N  
O  
T  
E  
S

- 1) General performance of API calls
  - API tracing provides microsecond timings in the trace record. By finding the API begin (i.e. MQGET >>) and the API end (i.e. reason field of MQGET <<) you can roughly calculate the time it took for the API MQGET to complete. Do note that tracing does add overhead to the timings.

```
48 13:15:37.742881 16884.1 CONN:1400006 MQOPEN >>
69 13:15:37.743138 16884.1 CONN:1400006 MQOPEN <<
88 13:15:37.743157 16884.1 CONN:1400006 Reason:
89 13:15:37.743158 16884.1 CONN:1400006 0x0000: 00000000
```

13:15:37.743158 - 13:15:37.742881 = 0.000277 seconds to complete for the MQOPEN

- mqapitrstats tool in the MH06 Trace Tools supportpac will read an entire API trace and create a summary report of the response times of the open, close, get, put, and put1 API calls. Executables are provided for Linux x86, Solaris Sparc, and Windows.

## MQ Tracing – Other Uses

N 2) Investigation of triggering issues

```
strmqtrc -m qmgr -t all -p runmqtrm
```

O ■ The runmqtrm trace will record if/when the trigger message was read from the INITQ, if/when it started the application of your process, and the operating system return code from the start call. Also, this does not require that runmqtrm be run in the foreground.

T

E

S



## Some Final MQ Tracing Notes

N

- Client applications can be traced, as well. You can either run a client trace on the client server (unfortunately, Java clients do not support this type of tracing) or trace the queue manager process that the SVRCONN channel is running on.

O

- Examples of client traces

T

1. `strmqtrc -t api -p prog1` (from client server)
2. `strmqtrc -m qmgr -t api -p amqrmppa` (from queue manager server)

E

S

## Some Final MQ Tracing Notes – cont

- N  
O  
T  
E  
S
- Tracing adds performance overhead and can create large files. Be judicious on the length of time that you run the trace and try and be selective with the options (i.e. `-t api -p prog1`) to reduce any unneeded output. Also, keep an eye on the size of your trace files and your space available on your trace file system (i.e. `/var/mqm`). You can also use the `strmqtrc -l` (MaxSize in MB) option to limit the size of your trace files, but this means that trace data can be overwritten and lost. The `-l` option keeps a current `AMQppppp.qq.TRC` and a previous `AMQppppp.qq.TRS` file.

```
strmqtrc -m qmgr -t api -p amqspout -l 1
```

- APAR IT01972 – Queue Manager trace is inadvertently turned off for an application thread with multiple shared connections after an MQDISC is called. End result is the potential for trace data loss. Targeted delivery of PTF is 7.1.0.6, 7.5.0.5, 8.0.0.1.

## Application Activity Trace (AAT)

- |   |                                                                                                                                                                                                                      |
|---|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| N | ▪ The Application Activity Trace (AAT) was first introduced in 7.1. It provides detailed information of the behavior of applications connected to a queue manager, including their MQI call details.                 |
| O | ▪ “Increasing the visibility of messages using WebSphere MQ Application Activity Trace” by Emma Bushby is an IBM DeveloperWorks article that does a good job in explaining the Application Activity Trace in detail. |
| T | ▪ The AAT is another tool that can be helpful in MQ problem determination or application review, by giving you visibility to the inputs and outputs of your application API calls.                                   |
| E |                                                                                                                                                                                                                      |
| S |                                                                                                                                                                                                                      |

## AAT – Usage Notes

- N
- Applications write AAT records to the SYSTEM.ADMIN.TRACE.ACTIVITY.QUEUE.
- O
- There is a hierarchy to turning ON/OFF the AAT:
    1. ACTVTRC queue manager attribute (ON/OFF) (overridden by)
    2. MQCNO\_ACTIVITY\_TRACE connection options specified in an MQCONN (NOTE: ACTVCONO queue manager attribute must be ENABLED for this to be checked, and the default value is DISABLED) (overridden by)
    3. Settings in a matching stanza in mqat.ini (located in qm.ini directory)
- T
- E
- In order to pick up a mqat.ini change dynamically in a running program, you need to toggle the ACTVTRC queue manager attribute (i.e. ON/OFF).
- S

## AAT – Viewing the Data

- N
- MS0P supportpac (WebSphere MQ Explorer Extended Management Plugins) has an Application Activity Trace viewer.
- O
- amqsact is a command line tool (sample code also provided) that can read the messages from the SYSTEM.ADMIN.TRACE.ACTIVITY.QUEUE and format them into summary and verbose reports.
- T
- amqsactz on Capitalware's Sample WebSphere MQ C Code web site is a program that takes the amqsact sample code and provides the following enhancements:
- E
1. Includes more data (i.e. Conn, Channel, etc.) on API one line summaries.
  2. Includes -r option for helpful summary reports
  3. Corrects a print formatting issue where a byte like x'DF' was printed as x'FFFFFFDF'.
- S

# AAT – amqsputc example

**N** 1) Add an ApplicationTrace stanza for amqsputc to the mqat.ini to turn on AAT tracing.

**O**

**T**

**E**

**S**

```
ApplicationTrace: # Application specific settings stanza
 ApplClass=ALL # Application type
 # Values: (USER | MCA | ALL)
 # Default: USER
 ApplName=amqsputc # Application name (may be wildcarded)
 # (matched to app name without path)
 # Default: *
 Trace=ON # Activity trace switch for application
 # Values: (ON | OFF)
 # Default: OFF
 ActivityInterval=0 # Time interval between trace messages
 # Values: 0-99999999 (0=off)
 # Default: 0
 ActivityCount=0 # Number of operations between trace msgs
 # Values: 0-99999999 (0=off)
 # Default: 0
 TraceLevel=MEDIUM # Amount of data traced for each operation
 # Values: LOW | MEDIUM | HIGH
 # Default: MEDIUM
 TraceMessageData=0 # Amount of message data traced
 # Values: 0-104857600
 # Default: 0
```

## AAT – amqsputc example

N 2) Run the amqsputc sample program.

```
mqm$ export MQSERVER='CLIENT.TO.SERVER/TCP/SERVER01'
mqm$ amqsputc TCZ.TEST1
Sample AMQSPUT0 start
target queue is TCZ.TEST1
test1
test2
test3
test4
test5
T Sample AMQSPUT0 end
```

E 3) Update ApplicationTrace stanza for amqsputc in the mqat.ini to turn off AAT tracing.

S NOTE: If instead, amqsputc was to continue to run and you turned the trace off with the mqat.ini change, you would need to toggle the ACTVTRC queue manager attribute ON/OFF to have amqsputc pick up the mqat.ini change.

## AAT – amqsactz reports

- N 4) Use amqsactz to view AAT data, by generating 3 reports:
- O 1. amqsactz.out – non-verbose report with –r summary information
- O 2. amqsactz\_1LS.out – API one line summaries selected from amqsactz.out
- T 3. amqsactv\_v.out – verbose report
- T ■ amqsactz.out – non-verbose report, includes –r summary output at bottom of report

```
amqsactz -r -b > amqsactz.out
```

E

S



# AAT - amqsactz.out

N  
O  
T  
E  
S

```
MonitoringType: MQI Activity Trace RecordNum: 0 ← 1
Correl_id:
00000000: 414D 5143 5345 5256 5245 3031 2E4D 5154 'AMQCSERVER01.MQT'
00000010: 53E3 C453 010A F420 'S..S...'
QueueManager: 'SERVER01.MQTEST1'
Host Name: 'server01'
IntervalStartDate: '2014-08-28'
IntervalStartTime: '09:24:29'
IntervalEndDate: '2014-08-28'
IntervalEndTime: '09:24:35'
CommandLevel: 750
SeqNumber: 0
ApplicationName: 'amqsputc' ← 2
Application Type: MQAT_UNIX
ApplicationPid: 24912 ← 3
UserId: 'mqm'
API Caller Type: MQXACT_EXTERNAL
API Environment: MQXE_MCA_SVRCONN
Channel Name: 'CLIENT.TO.SERVER' ← 4
ConnName: '127.0.0.1'
Channel Type: MQCHT_SVRCONN
Application Function: ''
Appl Function Type: MQFUN_TYPE_UNKNOWN
Trace Detail Level: 2
Trace Data Length: 0
Pointer size: 8
Platform: MQPL_UNIX
```

# AAT - amqsactz.out

N

O

T

E

S

UserId: 'mqm'  
API Caller Type: MQXACT\_EXTERNAL  
API Environment: MQXE\_MCA\_SVRCONN  
Channel Name: 'CLIENT.TO.SERVER'  
ConnName: '127.0.0.1'  
Channel Type: MQCHT\_SVRCONN  
Application Function: ''  
Appl Function Type: MQFUN\_TYPE\_UNKNOWN  
Trace Detail Level: 2  
Trace Data Length: 0  
Pointer size: 8  
Platform: MQPL\_UNIX

| EYEC        | RecordNum | Pid   | Tid  | Conn             | Channel Name     | Date       | Time     | Operation  | MQRC | HObj |
|-------------|-----------|-------|------|------------------|------------------|------------|----------|------------|------|------|
| (ObjName)   |           |       |      |                  |                  |            |          |            |      |      |
| 1LS= 4      |           | 24912 | 2606 | 53E3C453010AF420 | CLIENT.TO.SERVER | 2014-08-28 | 09:24:29 | MQXF_CONNX | 0000 | -    |
| 1LS= 0      |           | 24912 | 2606 | 53E3C453010AF420 | CLIENT.TO.SERVER | 2014-08-28 | 09:24:29 | MQXF_OPEN  | 0000 | 2    |
| (TCZ.TEST1) |           |       |      |                  |                  |            |          |            |      |      |
| 1LS= 0      |           | 24912 | 2606 | 53E3C453010AF420 | CLIENT.TO.SERVER | 2014-08-28 | 09:24:31 | MQXF_PUT   | 0000 | 2    |
| (TCZ.TEST1) |           |       |      |                  |                  |            |          |            |      |      |
| 1LS= 0      |           | 24912 | 2606 | 53E3C453010AF420 | CLIENT.TO.SERVER | 2014-08-28 | 09:24:31 | MQXF_PUT   | 0000 | 2    |
| (TCZ.TEST1) |           |       |      |                  |                  |            |          |            |      |      |
| 1LS= 0      |           | 24912 | 2606 | 53E3C453010AF420 | CLIENT.TO.SERVER | 2014-08-28 | 09:24:32 | MQXF_PUT   | 0000 | 2    |
| (TCZ.TEST1) |           |       |      |                  |                  |            |          |            |      |      |
| 1LS= 0      |           | 24912 | 2606 | 53E3C453010AF420 | CLIENT.TO.SERVER | 2014-08-28 | 09:24:33 | MQXF_PUT   | 0000 | 2    |
| (TCZ.TEST1) |           |       |      |                  |                  |            |          |            |      |      |
| 1LS= 0      |           | 24912 | 2606 | 53E3C453010AF420 | CLIENT.TO.SERVER | 2014-08-28 | 09:24:35 | MQXF_PUT   | 0000 | 2    |
| (TCZ.TEST1) |           |       |      |                  |                  |            |          |            |      |      |
| 1LS= 0      |           | 24912 | 2606 | 53E3C453010AF420 | CLIENT.TO.SERVER | 2014-08-28 | 09:24:35 | MQXF_CLOSE | 0000 | 2    |
| (TCZ.TEST1) |           |       |      |                  |                  |            |          |            |      |      |
| 1LS= 0      |           | 24912 | 2606 | 53E3C453010AF420 | CLIENT.TO.SERVER | 2014-08-28 | 09:24:35 | MQXF_DISC  | 0000 | -    |

# AAT - amqsactz.out (-r summary option)

N  
O  
T  
E  
S

```
=====
Application Summary Report
=====
Pid ApplicationName UserId Tid Count MQI Count
24912 amqsputc mqm 1 11

=====
Application Objects Referenced Report
=====
pid: 24912 ApplicationName: amqsputc UserId: mqm referenced the following objects:
ObjName: TCZ.TEST1 Count: 7

=====
Application Objects Options Report
Options tracked are conn, open, get, put, close, callback, sub, subrg
=====
pid: 24912 ApplicationName: amqsputc UserId: mqm referenced the following options by object:
Object Name: TCZ.TEST1
Open Options: 8208 Count: 1
MQOO_OUTPUT
MQOO_FAIL_IF QUIESCING
Put Options: 8260 Count: 5
MQPMO_NO_SYNCPOINT
MQPMO_NEW_MSG ID
MQPMO_FAIL_IF QUIESCING
Close Options: 0 Count: 1
MQCO_NONE
MQCO_IMMEDIATE
```

# AAT - amqsactz.out (-r summary option)

N

```
=====
Application Channels Referenced Report
=====
pid: 24912 ApplicationName: amqsputc UserId: mqm referenced the following channels:
ChannelName: CLIENT.TO.SERVER Count: 11
=====
```

O

```
=====
Application Operations Executed Report
=====
pid: 24912 ApplicationName: amqsputc UserId: mqm executed the
following operations:
Operation: MQXF_BACK Count: 1
Operation: MQXF_CLOSE Count: 1
Operation: MQXF_CONNX Count: 1
Operation: MQXF_DISC Count: 2
Operation: MQXF_OPEN Count: 1
Operation: MQXF_PUT Count: 5
=====
```

T

E

S

# AAT - amqsactz.out (-r summary option)

N  
O  
T  
E  
S

```
=====
Application Operations Options Report
Options tracked are conn, open, get, put, close, callback, sub, subrq
=====
pid: 24912 ApplicationName: amqsputc UserId: mqm referenced the following options by operations:
 Operation: MQXF_CLOSE
 Close Options: 0 Count: 1
 MQCO_NONE
 MQCO_IMMEDIATE
 Operation: MQXF_CONNX
 Connect Options: 320 Count: 1
 MQCNO_HANDLE_SHARE_BLOCK
 MQCNO_SHARED_BINDING
 Operation: MQXF_OPEN
 Open Options: 8208 Count: 1
 MQOO_OUTPUT
 MQOO_FAIL_IF QUIESCING
 Operation: MQXF_PUT
 Put Options: 8260 Count: 5
 MQPMO_NO_SYNCPOINT
 MQPMO_NEW_MSG_ID
 MQPMO_FAIL_IF QUIESCING
```

# AAT – amqsactz reports

N  
O  
T  
E  
S

- Remember, each API summary line had a 1LS= eye catcher text in it.

```
=====
EYEC RecordNum Pid Tid Conn Channel Name Date Time Operation MQRC HObj
(ObjName)
1LS= 0 24912 2606 53E3C453010AF420 CLIENT.TO.SERVER 2014-08-28 09:24:29 MQXF_CONNX 0000 -
1LS= 0 24912 2606 53E3C453010AF420 CLIENT.TO.SERVER 2014-08-28 09:24:29 MQXF_OPEN 0000 2
(TCZ.TEST1)
```

- amqsactz\_1LS.out - Use the 1LS= to grep out all the API one line summary records into a report.

```
grep 1LS= amqsactz.out > amqsactz_1LS.out
```

# AAT - amqsactz\_1LS.out

N  
O  
T  
E  
S

|             | 1 (RecordNum) | 2 (Pid) | 3 (Tid)          | 4 (Conn)         | 5 (Channel)         |            |      |   |  |  |  |
|-------------|---------------|---------|------------------|------------------|---------------------|------------|------|---|--|--|--|
| 1LS= 0      | 24912         | 2606    | 53E3C453010AF420 | CLIENT.TO.SERVER | 2014-08-28 09:24:29 | MQXF_CONNX | 0000 | - |  |  |  |
| 1LS= 0      | 24912         | 2606    | 53E3C453010AF420 | CLIENT.TO.SERVER | 2014-08-28 09:24:29 | MQXF_OPEN  | 0000 | 2 |  |  |  |
| (TCZ.TEST1) |               |         |                  |                  |                     |            |      |   |  |  |  |
| 1LS= 0      | 24912         | 2606    | 53E3C453010AF420 | CLIENT.TO.SERVER | 2014-08-28 09:24:31 | MQXF_PUT   | 0000 | 2 |  |  |  |
| (TCZ.TEST1) |               |         |                  |                  |                     |            |      |   |  |  |  |
| 1LS= 0      | 24912         | 2606    | 53E3C453010AF420 | CLIENT.TO.SERVER | 2014-08-28 09:24:31 | MQXF_PUT   | 0000 | 2 |  |  |  |
| (TCZ.TEST1) |               |         |                  |                  |                     |            |      |   |  |  |  |
| 1LS= 0      | 24912         | 2606    | 53E3C453010AF420 | CLIENT.TO.SERVER | 2014-08-28 09:24:32 | MQXF_PUT   | 0000 | 2 |  |  |  |
| (TCZ.TEST1) |               |         |                  |                  |                     |            |      |   |  |  |  |
| 1LS= 0      | 24912         | 2606    | 53E3C453010AF420 | CLIENT.TO.SERVER | 2014-08-28 09:24:33 | MQXF_PUT   | 0000 | 2 |  |  |  |
| (TCZ.TEST1) |               |         |                  |                  |                     |            |      |   |  |  |  |
| 1LS= 0      | 24912         | 2606    | 53E3C453010AF420 | CLIENT.TO.SERVER | 2014-08-28 09:24:35 | MQXF_PUT   | 0000 | 2 |  |  |  |
| (TCZ.TEST1) |               |         |                  |                  |                     |            |      |   |  |  |  |
| 1LS= 0      | 24912         | 2606    | 53E3C453010AF420 | CLIENT.TO.SERVER | 2014-08-28 09:24:35 | MQXF_CLOSE | 0000 | 2 |  |  |  |
| (TCZ.TEST1) |               |         |                  |                  |                     |            |      |   |  |  |  |
| 1LS= 0      | 24912         | 2606    | 53E3C453010AF420 | CLIENT.TO.SERVER | 2014-08-28 09:24:35 | MQXF_DISC  | 0000 | - |  |  |  |
| 1LS= 1      | 24912         | 2606    | 53E3C453010AF420 | CLIENT.TO.SERVER | 2014-08-28 09:24:35 | MQXF_BACK  | 0000 | - |  |  |  |
| 1LS= 1      | 24912         | 2606    | 53E3C453010AF420 | CLIENT.TO.SERVER | 2014-08-28 09:24:35 | MQXF_DISC  | 0000 | - |  |  |  |

6 (ObjName) NOTE: ObjName is being line wrapped, and is on same line as MQXF\_CLOSE)

## AAT – amqsactz reports

N

O

T

E

S

- amqsactz\_v.out – verbose report for each AAT record

```
amqsactz -v -b > amqsactz_v.out
```



# AAT - amqsactz\_v.out

N  
O  
T  
E  
S

```
MonitoringType: MQI Activity Trace RecordNum: 0
MQI Operation: 2
 Operation Id: MQXF_PUT
 ApplicationTid: 2606
 OperationDate: '2014-08-28'
 OperationTime: '09:24:31'
 High Res Time: 1409235871018640
 Completion Code: MQCC_OK
 Reason Code: 0
 Hobj: 2
 Put Options: 8260 ← 1
 Msg length: 5
 Known_dest_count: 1
 Unknown_dest_count: 0
 Invalid_dest_count: 0
 Object_type: MQOT_Q
 Object_name: 'TCZ.TEST1' ← 2
 Object_Q_mgr_name: ''
 Resolved_Q_Name: 'TCZ.TEST1'
 Resolved_Q_mgr: 'SERVER01.MQTEST1'
 Resolved_local_Q_name: 'TCZ.TEST1'
 Resolved_local_Q_mgr: 'SERVER01.MQTEST1'
 Resolved_type: MQOT_Q
 Report Options: 0
 Msg_type: MQMT_DATAGRAM
 Expiry: -1 ← 3
 Format_name: 'MQSTR'
 Priority: -1 ← 4
 Persistence: 2 ←
 Msg_id:
00000000: 414D 5120 5345 5256 5245 3031 2E4D 5154 'AMQ SERVER01.MQT'
00000010: 53E3 C453 020A F420 'S..S...' ,
```

## Linux Internals - /proc file system

- N
- O
- T
- E
- S
- The /proc file system is a virtual file system that allows you access to internal kernel data.
  - /proc/pid/enviro will show you the environment variables for a pid when the process was created. NOTE: Later changes to the environment after the process is created are not reflected here. See handout #8.
  - /proc/pid/limits will show the user limits for a process. Later versions of the Linux kernel (I believe 2.6.32) allow you to dynamically update the limits for a running process. For example, you can increase the Max open files setting for a running queue manager process. See handout #9.
  - /proc/pid/fd will show the file descriptors for the process. You could see which files that the process has open, or how many files are open. See handout #10.

## Linux Commands - strace

N

- strace will trace the system calls (i.e. open, write, etc.) that a process is making. This can be helpful to see internally what a process is doing for problem determination. You need the proper security to strace a process. For processes running under the mqm id, you need to be the mqm id.

O

- Possible uses are:
  1. strace the start up of the queue manager to see if an exit is being invoked.
  2. strace an actively running process that seems to be hung.
  3. strace a problematic MQ command.

T

- Helpful pieces to search for in the strace output:
  1. file names
  2. programs being executed
  3. error messages being written

E

S

See handout #11.

## Questions & Answers

