SHARE Anaheim – March 12, 2014
Session 14911

CICS and Threadsafe

Exploiting the Open Transaction Environment

Ivan L. Gelb – ivan@gelbis.com
Russ Evans - russ@evansgroupconsulting.com
732-303-1333
Objectives

• Definitions
• Recommendations
• History of CICS Multitasking
• The Open Transaction Environment
• Making programs Threadsafe
• Exploiting the OTE
• OTE Performance Considerations
• Diagnosing Threadsafe Problems
• Addendums
Definitions

1. “A threadsafe program is one that does not modify any area of storage that can be modified by any other program at the same time, and does not depend on any area of shared storage remaining consistent between machine instructions.”

2. “A program defined as CONCURRENCY=THREADSAFE is one that will be allowed to run on an open TCB.”

3. “A threadsafe CICS command is one that is allowed to run under an open TCB. A non-threadsafe command is one that is not allowed to run under an open TCB.”
Recommendations

• Consider Threadsafe implications now to:
  – Lower the Total Cost of Ownership (TCO) of CICS applications
  – Enable automatic exploitation of latest future hardware and software capabilities
• Target the Heavy CPU users to exploit multiprocessors
• Review “behavior” of purchased software packages
• Beware of COBOL calls (dynamic or static)
Recommendations

- OTE exploitation project maybe a waste unless we:
  - Convert XRMIIN/OUT and Dynamic Plan Selection exits before migrating to a threadsafe capable CICS release
  - Convert all frequently used exit programs to threadsafe before converting programs
  - Verify that required maintenance is on CICS and vendor products before converting programs to threadsafe
  - Study the IBM Redbook “Threadsafe Considerations for CICS”
History of CICS Multitasking

• CICS as a Single TCB exploiter:
  – Most efficient on fastest single processor
  – “Quasi-Reentrancy” is not = Reentrant code
  – Issues:
    • Region capacity limited by speed of one processor even though many CPUs may be available
    • Forces “cloning” of regions to gain capacity
    • Runaway tasks hang the entire CICS region
    • OS Waits in CICS applications, not a recommended practice, cause Wait of the entire Region
    • Many restricted OS and COBOL Commands
History of CICS Multitasking

- CICS Exploitation of Multiple Processors
  - Multiple TCBs available, but
  - Quasi-Reentrant (QR) is only one used for majority of transactions processing
  - Additional TCBs for:
    - VSAM
    - DB2
    - Program Loader
    - etc.
    - Except for the DB2 activity, most of these activities consume very small amounts of CPU
History of CICS Multitasking

- CICS and DB2
  - Separate TCB (‘thread’) for each DB2 Request
  - Task is switched from QR to DB2 TCB
  - DB2 code runs on DB2 TCB
  - Significant workload may be shifted from QR to DB2 TCBs, but added measurable overhead for TCB back and forth switching
Open Transaction Environment (OTE)

- Transaction runs under own TCB
- Introduced in TS 1.3 for Java
- DB2 Support added for TS 2.2
- Supports full OS function
- Allows true Multitasking in CICS
- Pseudo-reentrancy no longer allowed
OTE and DB2

Without Threadsafe

**QR TCB**

Task Starts

EXEC CICS

EXEC SQL

Application Code

EXEC SQL

Open TCB

DB2 Code executes

DB2 Code completes

DB2 Code executes

DB2 Code completes

Copyright (c) 2014, The Evans Group, Inc. & Ivan L. Gelb
OTE and DB2

With Threadsafe

QR TCB
Task Starts
EXEC CICS
EXEC SQL
DB2 Code executes
Application Code
DB2 Code executes
Task Termination
Task completes
So, What’s the Problem

CICSRGN1

TASK1

PROG001

MOVE CWA-COUNTER TO OUTPUT-FIELD
ADD +1 TO CWA-COUNTER
EXEC CICS WRITE OUTPUT-RECORD

CWA

0001

0001 + 1 = 0002

stuff0001morestuff

Copyright (c) 2014, The Evans Group, Inc. & Ivan L. Gelb
So, What’s the Problem

CICSRGN1

TASK1

PROG001

MOVE CWA-COUNTER TO OUTPUT-FIELD
ADD +1 TO CWA-COUNTER
EXEC CICS WRITE OUTPUT-RECORD

stuff0001morestuff

0001 + 1 = 0002

CWA

0001

0002 + 1 = 0003

TASK2

PROG001

MOVE CWA-COUNTER TO OUTPUT-FIELD
ADD +1 TO CWA-COUNTER
EXEC CICS WRITE OUTPUT-RECORD

stuff0001morestuff

0001
Controlling Threadsafe

- At the region level, new SIT parm - FORCEQR=YES/NO:
  - FORCEQR=YES  All programs run on QR TCB
  - FORCEQR=NO Programs follow CONCURRENCY parm on program definition

- At the program level:
  New parameter on Program Definition
  - CONCURRENCY=QUASIRENT  (Not Threadsafe)
  - CONCURRENCY=THREADSAFE
  - CONCURRENCY=REQUIRED
Identifying Threadsafe Programs

• CONCURRENCY parameter is a “promise” by Developer, not an order to CICS
• No automated method of identification
• IBM Tool helps: DFHEISUP will scan for CICS commands commonly used in non-threadsafe applications
• Rules of thumb:
  – COBOL and PL/1 must be LE
  – All programs must be re-entrant
  – Aps with no affinities are more likely to be threadsafe
Identifying Threadsafe Programs

Ensure programs are re-entrant:

• COBOL:
  – Compile with RENT
  – Link with RENT

• Assembler:
  – Code review, possible coding changes required
  – Assemble/Link with Rent

• CICS:
  – RENTPGM=PROTECT
  – Adjust RDSA/ERDSA sizes
  – Non-reentrant activity will generate DFHSR0622 message followed by S0C4/ASRA abend
  – Possible conflicts with debuggers
IBM supplied tool available to help start.....

- Utility DFHEISUP will scan for CICS commands commonly used in non-threadsafe applications
- Use command table DFHEIDTH
Making Programs Threadsafe

After identifying non-Threadsafe code you have two choices (one requires applications changes):

1) Alter the code to serialize the shared storage access
   A) Use CICS to automatically ensure serialization
   B) Manually ensure serialization

2) Do nothing
If shared storage use is limited to few programs:

- Leave non-threadsafe programs QUASIRENT
- CICS will switch to QR on LINK or XCTL (But… *not for CALL commands!*)
- Access to shared storage is automatically serialized by CICS
CICSRGN1

TASK1

PROG001

MOVE CWA-COUNTER TO OUTPUT-FIELD
ADD +1 TO CWA-COUNTER
EXEC CICS WRITE OUTPUT-RECORD

stuff
0001 + 1 = 0002

CWA

0001

morestuff

TASK2

PROG001

Wait For QR

Making Programs Threadsafe

Leave Program CONCURRENCY(QUAISRENT)
Leave Program CONCURRENCY(QUASIRENT)

Advantages:
• No coding changes, so it can be a quick implementation

Disadvantages:
• Additional TCB switching overhead
• Maintenance issues
• All programs that access the shared storage areas **must** also remain QUASIRENT
• Applications mostly limited to process on QR TCB
To serialize access to shared storage:

- “Wrap” access in CICS ENQ/DEQ – and measure / track response time elongation this will cause
- For Assembler, use CS/CDS instructions
- Move data to threadsafe but serialized facility like:
  - CICS Maintained Data Table
  - DB2 table
  - Coupling Facility
Making Programs Threadsafe

With ENQ / DEQ

CICS RGN 1

**Task 1**

**Program 001**

```
EXEC CICS ENQ()
MOVE CWA-COUNTER TO OUTPUT-FIELD
ADD +1 TO CWA-COUNTER
EXEC CICS WRITE OUTPUT-RECORD
```

0001 + 1 = 0002

**Task 2**

**Program 001**

```
EXEC CICS ENQ()
...wait on enq...
MOVE CWA-COUNTER TO OUTPUT-FIELD
ADD +1 TO CWA-COUNTER
EXEC CICS WRITE OUTPUT-RECORD
```

0002 + 1 = 0003

Copyright (c) 2014, The Evans Group, Inc. & Ivan L. Gelb
making programs threadsafe continued...

**ENQ Issues:**

- CPU Cost relatively low but waits for ENQ can severely impact application response times as load increases
- Minimize the Potential bottleneck
  - Limit ENQ duration by issuing DEQ as soon as possible
  - Ensure deadly embrace prevention via strictly enforced application design.
    The “classic” safe design is preserved ENQ order in every program.
  - Prepare for deadly embraces with appropriate limiting parameter - DTIMEOUT
Regardless of which method, remember:

**All** programs that access the same shared storage areas in the same CICS region **must** be converted before **any** of these programs are marked as Threadsafe!
Accessing The OTE

Two supported methods of executing on OTE TCB

- Define program as API(OPENAPI)
- Define program as CONCURRENCY(REQUIRED)
Accessing The OTE via OPENAPI

For CICS 3.1 and higher, modify the PROGRAM definition on the application program to API=OPENAPI

- The program **must** be Threadsafe
- All application code runs in the OTE environment
- All application code runs on the same TCB instance on which the program was initialized.
Accessing The OTE via OPENAPI

API=OPENAPI Forces program to run on L8/9 TCB:

- Program is initialized on L8 TCB if CICS key
- Program is initialized on L9 TCB if USER key
- If program issues non-threadsafe command, task is switched to QR TCB
- Once non-threadsafe command completes, task is switched back to L8/9
- Use INQUIRE_CURRENT_PROGRAM and INQUIRE_PROGRAM to identify status

Copyright (c) 2014, The Evans Group, Inc. & Ivan L. Gelb
Accessing The OTE via OPENAPI

QR TCB

Open TCB

Task Starts

E.C. threadsafe

E.C. threadsafe

Command Starts

E.C. non-threadsafe

Command Completes

Task Termination

Copyright (c) 2014, The Evans Group, Inc. & Ivan L. Gelb
Accessing The OTE via OPENAPI

There are performance implications for USER key OPENAPI programs that also access OPENAPI TRUEs (includes DB2)

- USER key Program is initialized on L9 TCB
- OPENAPI TRUE is initialized on L8 TCB
- When L9 program issues DFHRMCAL to OPENAPI TRUE:
  - Task is switched to L8 TCB for duration of TRUE
  - Task is returned to L9 following completion of TRUE
- L8 TCB instance held until task termination
Accessing The OTE via OPENAPI

There are performance issues for USER key OPENAPI programs that also access OPENAPI TRUEs (includes DB2)

- Review MAXOPENTCB for possible increase
- Review TCBLIMIT for possible increase
- Open TCB “stealing” performance issues
- Potential TCB deadly embrace
Accessing The OTE with CONCURRENCE(REQUIRED)

For CICS 4.2, modify the PROGRAM definition on the application program to API(CICSAPI) and CONCURRENCE(REQUIRED)

- The program **must** be Threadsafe
- **All** application code runs in the OTE environment
- **All** application code runs on the same TCB instance on which the program was initialized.
- **All** application code runs on an L8 TCB
Accessing The OTE with CONCURRENCY(REQUIRED)

Forces program to run on L8 TCB:

• Program is initialized on L8 TCB
• If program issues non-threadsafe command, task is spun to QR
• Once command has completed, task is spun to L8
• Use INQUIRE_CURRENT_PROGRAM and INQUIRE_PROGRAM to identify
QR TCB

Command Starts

Open TCB

Task Starts

E.C. threadsafe

E.C. threadsafe

Command Completes

E.C. non-threadsafe

Task Termination
Accessing The OTE with CONCURRENCY(REQUIRED)

There are no additional performance issues / resource requirements for USER key CONCURRENCY(REQUIRED) programs that also access OPENAPI TRUEs (includes DB2)

- USER key Program is initialized on L8 TCB
- OPENAPI TRUE is initialized on L8 TCB
- Only one L8 TCB is acquired by the task
  - L8 is shared by user program and all OPENAPI TRUEs
- L8 TCB instance held until task termination
Accessing The OTE

So.......}

Which Way Is Best?
Accessing The OTE

Via OPENAPI Parm

Advantages:
• No coding changes required
• All application code **guaranteed** to run in OTE
• No requirement to enable TRUE
• Can determine environment programmatically
• All user code on same TCB – no issues with “paired” z/OS macros
Accessing The OTE

Via OPENAPI Parm

Disadvantages:
- CPU overhead when accessing OPENAPI TRUE in USER key (DB2, etc.)
- CPU overhead for TCB switching when issuing non-threadsafe EXEC CICS commands
- All application logic must be threadsafe
- Can increase the number of open TCBs required.
- Overhead if TCB stolen to switch key
Accessing The OTE

Via CONCURRENCY(REQUIRED) Parm

Advantages:
• No coding changes required
• All application code guaranteed to run in OTE
• No requirement to enable TRUE
• Can determine environment programmatically
• All user code on same TCB – no issues with use of “paired” z/OS macros
Accessing The OTE

Via CONCURRENCY(REQUIRED) Parm

Disadvantages:

• CPU overhead when issuing non-threadsafe EXEC CICS commands

• All application logic **must** be threadsafe
Accessing The OTE

Via CONCURRENCY(REQUIRED)
with
API(OPENAPI)

Disadvantages:
• Can increase the number of open TCBs required.
• Overhead if TCB stolen to switch key
Accessing The OTE

Via CONCURRENCY(REQUIRED) with API(CICSAPI)

Disadvantages:

• Limited to using standard CICS services which is recommended practice anyway
• Potential for “unusual” and undocumented problems if unsupported z/OS services used
Accessing The OTE

One restriction in OPENAPI programs:

- Do not attempt to initialize batch LE environment under CICS OPENAPI.
Implications of New TCB Types

• Multiple TCB types
• Application code running in OTE
  – Application programs fighting for CPU
  – Poor coding only affects program user, not region
  – Resource hogs impact minimized
• CICS system code running in multiple TCBs
• IBM converting sub-products to use OTE
  – MQ
  – Sockets
  – XML parser
Multiple TCB Structure

Classic CICS

z/OS

CPU1
CICS/QR
Task1

CPU2
CICS/L8
DB2

CPU3
z/OS
Sockets

CPU4
MQ

Copyright (c) 2014, The Evans Group, Inc. & Ivan L. Gelb
Multiple TCB Structure

Modern CICS

z/OS

CPU1
CICS/QR
Task1

CPU2
CICS/L8
DB2

CPU3
CICS/L8
CICS
Sockets

CPU4
CICS/L8
CICS
XML

CICS/X8
C++

CICS/J8
Java

Copyright (c) 2014, The Evans Group, Inc. & Ivan L. Gelb
Why Bother?

Run tasks on an open TCB to:

• Reduce QR CPU constraint by running tasks concurrently on multiple TCB/multiple processors, and thus eliminate some clone AORs to save CPU and real storage use
• Segregate troublesome transactions to minimize their effects
• Use z/OS functionality forbidden on QR TCB (NOT a recommended practice for CICS applications) such as:
  – Activity generating z/OS waits
    • I/O
    • ENQ/DEQ
    • z/OS WAIT
Reducing QR CPU Constraint

QR TCB is limited to the speed of one processor

When QR hits CPU limit, region stalls

• Classic fixes
  ➢ Clone Region to offload CPU

• Modern fix = Exploit OTE to offload CPU

• Classic fix has higher CPU and real storage cost
Reducing QR CPU Blocking

QR TCB is single threaded

- Current task “owns” QR until next EXEC CICS (*)
- Heavy CPU routines don’t release QR
- Region appears to lock up
- While task runs, CICS workload backs up
  - VSAM, DB2 I/O Completes
  - New tasks ready for dispatch
  - .....
Reducing QR CPU Blocking

OTE is Multi-Threaded

• OTE task “owns” his TCB until next EXEC CICS (*)
• QR is available for other workload
• No region hold-up
• No extended response times
  • Other workload unaffected
  • Response time improves
Reducing QR CPU Constraint

Warning: Consider LPAR CPU Implications when converting a QR constrained region to exploit open TCBs:

- Reduce QR constraint by moving tasks to other processors
- In MP environment, total CPU will increase until:
  1. CICS CPU requirements satisfied
  2. Box CPU capacity met

- Can negatively impact z/OS workload CICS may depend on
Using Forbidden Functionality

Use almost any z/OS function (this is not a recommendations to do so):

- OTE provides insulation from demands of processor intensive transactions
  - CPU intensive tasks don’t monopolize QR TCB
  - QR available for CEMT, etc.
- Communicate with operator via WTOR
- Make use of flexibility of STORAGE OBTAIN/RELEASE
- Issue I/O without CICS file control
- Use z/OS ENQ/DEQ to synchronize with batch jobs
- Etc..etc………. 
OTE Performance Considerations

There are several performance issues that are unique to the OTE:

- Non-Threadsafe EXEC CICS commands
- Non-Threadsafe CICS Global User Exits
- Multi-TCB issues with OPENAPI programs
Non-Threadsafe CICS Commands

- Many commands not Threadsafe
- Use of non-Threadsafe commands *is fully supported* by CICS
- CICS detects non-threadsafe command and switches task to QR TCB
- Task’s TCB status following command depends on API definition
- Potential performance issue for API=OPENAPI
Non-Threadsafe CICS Commands

A list of the commands that are threadsafe can be found in the *CICS Application Programming Reference Manual*, under **CICS threadsafe commands in the API**.

A list of the threadsafe SPI commands can be found in the *CICS System Programming Reference Manual*, in Appendix D, **Threatsafe SPI commands**
Non-Threadsafe CICS Exits

• Significant area of concern
• Task switched to QR for duration of exit, then back to Open TCB
• Infrequently referenced exits less of a problem
• Frequently referenced exits (eg., XEIIN) are a major performance problem
• XRMIIN/OUT and Dynamic Plan Selection most worrisome
• Worst case: significant (20%++?) increase in CPU utilization.
• Can cause CPU impact even if FORCEQR=YES
Non-Threadsafe CICS Exits

- Use DFH0STAT to identify exits in use
  - Select DB2, User Exit and Global User Exit options
  - Identifies all active exits by program name, CONCURRENCY option, exit point, and GWA usage
  - Shows Dynamic Plan exits
- Identify vendor exits and contact vendor
  - Do not mark threadsafe without vendor OK
  - Do not convert with heavily used QUASIRENT exits
- Review homegrown exit code to ensure threadsafe
- Use IBM supplied utility, DFH$MOLS, to analyze SMF 110 records
Minimizing CPU Overhead

• CPU overhead caused by task switching is incurred when a non-Threadsafe command is issued while the task is running on an Open TCB
• Overhead is zero if no non-Threadsafe commands are issued while the task is running on an Open TCB
• Overhead is minimized when non-Threadsafe commands can be clustered on the QR thus minimizing task switching

EXEC SQL OPEN CURSOR
PERFORM UNTIL ...
   EXEC SQL FETCH....
   EXEC CICS WRITEQ TD
END-PERFORM
Minimizing CPU Overhead

Once the command has been identified…..

• Replace: Replace Transient Data with CICS TempStor?

OR

• Relocate: Move the command outside of the SQL loop?
Minimizing CPU Overhead

Replace Transient Data with CICS Temporary Storage:

EXEC SQL OPEN CURSOR
PERFORM UNTIL ...
   EXEC SQL FETCH....
   EXEC CICS WRITEQ TS
END-PERFORM
Minimizing CPU Overhead

QR TCB
Task Starts
FETCH

Open TCB
DB2 Code executes
WRITEQ TS
FETCH
WRITEQ TS
Minimizing CPU Overhead

Relocate Transient Data Writes:
EXEC SQL OPEN CURSOR
PERFORM UNTIL ...
  PERFORM VARYING...
    EXEC SQL FETCH...
    MOVE RESULTS TO WS-RESULTS()
  END-PERFORM
PERFORM VARYING...
  EXEC CICS WRITEQ TD FROM(WS-RESULTS())
  END-PERFORM
END-PERFORM
Minimize OTE Overhead: OPENAPI Program

CPU overhead is minimized when:

1. Program does not issue Non-Threadsafe commands

2. If USER key, no DB2 or OPENAPI TRUE calls issued by the program
Minimize OTE Overhead: OPENAPI Program

MQ Series With OPENAPI program in USER key

<table>
<thead>
<tr>
<th>L9 TCB</th>
<th>L7 TCB</th>
<th>QR TCB</th>
<th>MQ TCB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task Starts</td>
<td>EXEC SQL</td>
<td>DB2 code executes</td>
<td>DB2 code complete</td>
</tr>
<tr>
<td>E.C. WRITEQ TD</td>
<td>WRITEQ TD starts</td>
<td>WRITEQ TD ends</td>
<td></td>
</tr>
<tr>
<td>MQ PUT</td>
<td></td>
<td>MQ code executes</td>
<td>MQ code complete</td>
</tr>
<tr>
<td>E.C. RETURN</td>
<td></td>
<td>Task termination</td>
<td></td>
</tr>
</tbody>
</table>

Copyright (c) 2014, The Evans Group, Inc. & Ivan L. Gelb
MQ Series With OPENAPI program in CICS key

L9 TCB  L8 TCB  QR TCB  MQ TCB
Unused

DMYTRUE executes
Threadsafe code
EXEC SQL
E.C. WRITEQ TD

WRITEQ TD starts
WRITEQ TD ends
MQ PUT
MQ code executes

Task Starts
CALL ‘DMYRMCAL’

MQ code complete
Task termination

Copyright (c) 2014, The Evans Group, Inc. & Ivan L. Gelb
Minimize OTE Overhead: OPENAPI Program
Relocation Ineffective for OPENAPI!

QR TCB

Open TCB

Task Starts
OTE user code

WRITEQ TS

Inner Loop

WRITEQ TD
WRITEQ TD
WRITEQ TD
WRITEQ TD

Outer Loop

Copyright (c) 2014, The Evans Group, Inc. & Ivan L. Gelb
Minimize OTE Overhead: REQUIRED Program with API(CICSAPI)

CPU overhead is minimized when:

1. Program does not issue Non-Threadsafe commands
Minimize OTE Overhead: REQUIRED Program
Relocation Ineffective for REQUIRED!

QR TCB

Open TCB

Task Starts
OTE user code

WRITEQ TS

Inner Loop

WRITEQ TD

WRITEQ TD

WRITEQ TD

WRITEQ TD

Outer Loop
Reducing CPU Overhead

- Prior to CICS 4.2, IRC is not threadsafe. Threadsafe commands that are function shipped will be treated as if they are non-threadsafe.

- As of CICS 4.2 IPIC connections support threadsafe mirror transactions
Ensuring Threadsafe Coding When Creating New Programs

Design is critical

- Design goal = Minimize number of TCB switches / transaction
- Ensure threadsafe coding standards are met
Ensuring Threadsafe Coding When Creating New Programs

Ensure Threadsafe Coding Standards

- Eliminate updates to shared storage areas:
  - CWA
  - GWA
  - GETMAIN(SHARED)
  - OS GETMAIN
  - LOAD HOLD
- Require use of RENT on link-edit step
- Use RENTPGM=PROTECT in CICS
Ensuring Threadsafe Coding When Creating New Programs

Minimize number of TCB switches

- Insures best possible performance
- Use only Threadsafe commands
- Design program flow to cluster OTE usage
- Issue non-Threadsafe commands before or after OTE activity complete
Diagnosing Threadsafe Problems

No way to prove threadsafe!

• Threadsafe problems most likely to occur during peak time.
• Stress testing must mimic production to bring out threadsafe problems.
• Best way to ensure success is strong application knowledge.
• Be thorough in your review.
Diagnosing Threadsafe Problems

How to tell when Testing is Complete?

Hint: 100% solution unavailable

- Errors based on probability
- Difficult to force simultaneous execution of code path
- Use stress testing
  - Set MAXTASK high
  - Set DSALIMITs high
  - Set SYSDUMPING on!
  - Use driver program to issue large number of STARTs
Unpredictable Results Means Just That!

- Difficult to identify root cause
- “Impossible/unpredictable” behavior likely to be threadsafe issue
- Use CICS Auxtrace – be selective or you will be overloaded with useless details
- Consider homegrown application trace analysis code
- Last resort: CICS system dumps
Diagnosing Threadsafe Problems

Paired MVS macros that need same TCB

- Macros such as ENQ and DEQ must run on same TCB
- Intervening user code can force TCB switch
- **If switched, Second macro in pair fails**
- Macros include:
  - ENQ/DEQ
  - ATTACH/DETACH
Diagnosing Threadsafe Problems

A Statically Called Assembler Program Isn’t Threadsafe
due to self contained register save area

ASMPGM1 CSECT
LA R13, SAVEAREA
STM R14, R12, 12(R13)
LM R14, R12, 12(R13)
BR R14
SAVEAREA DS 18F

COBPGM
CALL ‘ASMPGM1’
USING PARM-LIST.
Diagnosing Threadsafe Problems

All Called Routines Run on TCB of the Caller

• Because ASMPGM1 issues no CICS commands, the code runs normally in a non-threadsafe environment
• CICS is not notified for calls
• Simultaneous access to SAVEAREA results in overlay
• Probable S0C4
• Identifiable in test via RENTPGM=PROTECT
Diagnosing Threadsafe Problems

All Called Routines Run on TCB of the Caller

Possible solutions:

1. Convert ASMPGM1 to Command Level
2. Alter COBPGM to pass address of RSA
3. Leave COBPGM non-Threadsafe
4. Convert ASMPGM1 to LE enabled Assembler
Threadsafe File Control

Threadsafe VSAM RLS available with CICS 3.2
Threadsafe local VSAM shipped in CICS 3.2 as disabled
New SIT parm:

\[
FCQRONLY=[\text{YES} \mid \text{NO}]
\]

- FCQRONLY=YES forces all file control to run on QR TCB
- FCQRONLY=NO allows threadsafe file control requests to run on L8/L9 TCB

Remote VSAM on non-IPIC connections remains non-threadsafe
Threadsafe File Control

Enable local VSAM threadsafe in CICS 3.2 with PTF UK37688
VSAM APARs OA20352 and OA24071 are required

NOTE: UK37688 changes the default on FCQRONLY from NO to YES. If you are running VSAM RLS threadsafe, and take the default on FCQRONLY, applying UK376688 will disable RLS threadsafe.
Futures

“It is the intention of IBM for future releases of CICS Transaction Server for z/OS to continue to enhance OTE support to enable the ongoing migration of CICS and application code from the QR to open TCBs.”
Futures

• IBM committed to making more commands threadsafe
• IBM Announced additional threadsafe commands in every release since TS 2.2
• CICS 3.2 introduces threadsafe file control (local)
  Note: CICS TS 3.2 was shipped with threadsafe VSAM disabled. Apply PK45354 to activate it
• CICS 4.2 introduced threadsafe DBCTL for DLI
• Conversion to OPENAPI TRUEs for CICS Sockets, MQ
• Internal use of OPENAPI for CPU intensive processes
Be @Next SHARE / Any Questions?

Join us at the next SHARE in Pittsburgh, Summer of 2014
Addendums

• Using “Forbidden” Functionality
• DFH$MOLS
• Minimizing Overhead
Using Forbidden Functionality

Transaction initiated communication with operator via WTOR:

- OTE TCB waits, not entire region
- Synchronous waits on external events/requests
- CICS command input from master console
- Enable use of standard auto operation facility

Disadvantages:
- Task shows as “running”
- No way to track WTOR back to task
Using Forbidden Functionality

Use of z/OS STORAGE OBTAIN/RELEASE

- Powerful options not available from EXEC CICS GETMAIN
- Storage acquired outside of CICS subpools
- More efficient than CICS GETMAIN

Disadvantages:

- Storage invisible to CICS monitor
- No automatic cleanup at task termination
- Storage not displayed in dump, trace, etc.
- Problems with OS GETMAIN and USER key OPENAPI tasks
Using Forbidden Functionality

Error on STORAGE OBTAIN causes ASRB, not region failure:

DFHAP0001 CICSD225 An abend (code 878/AKEB) has occurred at offset X'FFFFFFFF' in module TEST.

00057 L9002 AP 00E1 EIP EXIT LOAD
00057 L9002 AP 1942 APLI *EXC* Abend
00057 L9002 AP 0791 SRP *EXC* MVS_ABEND
00057 L9002 DS 0010 DSBR ENTRY INQUIRE_TASK
00057 L9002 DS 0011 DSBR EXIT INQUIRE_TASK/OK
00057 QR PG 0500 PGIS ENTRY INQUIRE_CURRENT_PROGRAM
00057 QR PG 0501 PGIS EXIT INQUIRE_CURRENT_PROGRAM
00057 QR AP 0782 SRP *EXC* ABEND_ASRB

TCB is marked as unusable:

DSTCB QR KE 0502 KEDS ENTRY DETACH_TERMINATED_OWN_TCBS
DSTCB QR KE 0503 KEDS EXIT DETACH_TERMINATED_OWN_TCBS/OK
Using Forbidden Functionality

Issue I/O without CICS file control:
• Bypass CICS file control
• “Batch-like” transactions segregated from normal CICS processing

Disadvantages:
• Cannot issue OPEN/CLOSE in COBOL program
• No backout or forward recovery
• Activity not in dump, trace, etc.
Using Forbidden Functionality

Reminder: the OTE only supports CICS LE service routines:

- COBOL display becomes a WRITEQ TD (not threadsafe!)
- COBOL dynamic call modified for CICS
- OPEN/CLOSE unavailable
- Storage obtained via EXEC CICS GETMAIN
Using IBM Utility DFH$MOLS

• IBM supplied utility, DFH$MOLS to analyze SMF 110 records
• Provides detailed report
  – One page / task
  – Storage utilization
  – CPU utilization
    • By TCB type
  – Response time
• Can use pre-generated MCT A$
• Activate monitoring with CEMT
  – SET MON ON PER
• Flush buffers with CEMT
  – SET MON ON NOP
Using IBM Utility DFH$MOLS

Use IFASMFDP to extract the 110 records

```
//*********************************************************
//* Step 1: Unload data from the SMF data sets
//*********************************************************
//SMFDUMP  EXEC PGM=IFASMFDP
//INDD1     DD DSN=SYS1.D002.MAN11,DISP=SHR,AMP=('BUFSP=65536')
//INDD2     DD DSN=SYS1.D002.MAN12,DISP=SHR
//INDD3     DD DSN=SYS1.D002.MAN13,DISP=SHR
//OUTDD1    DD DSN=?????.SMF.DATA1,DISP=(NEW,CATLG),
//            SPACE=(CYL,(50,10)),UNIT=SYSDA
//SYSPRINT  DD SYSOUT=A
//SYSIN     DD *
```

INDDx points to your SMF datasets. You can use either active datasets or archives.

OUTDD1 points to the output dataset that holds the extracted 110 records.

The OUTDD control statement describes your output file and the record types to be extracted. We’re using 110 subtype 1 records.

Use an INDD control statement to describe each SMF file used as input.
Using IBM Utility DFH$MOLS

Use DFH$MOLS to format the extracted records

```
//PRNT EXEC PGM=DFH$MOLS
//STEPLIB DD DSN=SYS2.CICSTS31.CICS.SDFHLOAD,DISP=SHR
//INPUT DD DSN=?????.SMF.DATA1,DISP=OLD
//SORTWK01 DD SPACE=(CYL,(5,1)),UNIT=SYSDA
//SORTWK02 DD SPACE=(CYL,(5,1)),UNIT=SYSDA
//SORTWK03 DD SPACE=(CYL,(5,1)),UNIT=SYSDA
//SORTWK04 DD SPACE=(CYL,(5,1)),UNIT=SYSDA
//SORTWK05 DD SPACE=(CYL,(5,1)),UNIT=SYSDA
//SORTDIAG DD SYSOUT=A
//SYSPRINT DD SYSOUT=A
//SYSOUT DD SYSOUT=A
//SYSPRINT DD SYSOUT=A
//SYSABEND DD SYSOUT=A
//SYSUDUMP DD SYSOUT=A
//SYSIN DD *
SELECT TRANID=trn1,trn2
DATE START=03/23/2006
/*
```

Use the SELECT TRANID cards to limit your report.

Use the DATE START card to limit your report.

The report is written to SYSPRINT.

INPUT DD points to OUTDD dataset from previous step.
<table>
<thead>
<tr>
<th>FIELD-NAME</th>
<th>UNINTERPRETED</th>
<th>INTERPRETED</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFHTASK C001</td>
<td>TRAN</td>
<td>C5E2C3F1</td>
</tr>
<tr>
<td>DFHTERM C002</td>
<td>TERM</td>
<td>C3D7F8F4</td>
</tr>
<tr>
<td>DFHCICS C089</td>
<td>USERID</td>
<td>C3C9C3E2 C4F2F2F4</td>
</tr>
<tr>
<td>DFHTASK C004</td>
<td>TTYPE</td>
<td>E3D60000</td>
</tr>
<tr>
<td>DFHCICS T005</td>
<td>START</td>
<td>BED82B7ADC91D761</td>
</tr>
<tr>
<td>DFHCICS T006</td>
<td>STOP</td>
<td>BED82B7ADD3A7B40</td>
</tr>
<tr>
<td>DFHTASK P031</td>
<td>TRANNUM</td>
<td>0000513C</td>
</tr>
<tr>
<td>DFHTASK A109</td>
<td>TRANPRI</td>
<td>00000001</td>
</tr>
<tr>
<td>DFHTERM C111</td>
<td>LUNAME</td>
<td>E2F0F1E3 C3D7F8F4</td>
</tr>
<tr>
<td>DFHPROG C071</td>
<td>PGMNAME</td>
<td>C5E2D7E4 E2C5C3F1</td>
</tr>
<tr>
<td>DFHTASK C097</td>
<td>NETUOWPX</td>
<td>C2C8C4D5 C5E34BE2 F0F1E3C3 D7F8F400 00000000</td>
</tr>
<tr>
<td>DFHTASK C098</td>
<td>NETUOWSX</td>
<td>D82B7ADC9D100001</td>
</tr>
<tr>
<td>DFHCICS A131</td>
<td>PERRECNT</td>
<td>00000001</td>
</tr>
<tr>
<td>DFHTASK T132</td>
<td>RMUOWID</td>
<td>BED82B7ADC9D1021</td>
</tr>
<tr>
<td>DFHCICS C167</td>
<td>SRVCLSNM</td>
<td>C3C9C3E2 40404040</td>
</tr>
<tr>
<td>DFHTASK C163</td>
<td>FCTYNAME</td>
<td>C3D7F8F4</td>
</tr>
<tr>
<td>DFHTASK A164</td>
<td>TRANFLAG</td>
<td>4000800002000000</td>
</tr>
<tr>
<td>DFHTASK A165</td>
<td>TERMINFO</td>
<td>01000191</td>
</tr>
<tr>
<td>DFHTASK C082</td>
<td>TRNGRPID</td>
<td>180FC2C8C4D5C5E3...</td>
</tr>
<tr>
<td>DFHTERM C197</td>
<td>NETID</td>
<td>C2C8C4D5 C5E34040</td>
</tr>
<tr>
<td>DFHTERM C198</td>
<td>RLUNAME</td>
<td>E2F0F1E3 C3D7F8F4</td>
</tr>
</tbody>
</table>

Copyright (c) 2014, The Evans Group, Inc. & Ivan L. Gelb
Non-Threadsafe CICS Exits

DFH$MOLS report of non-threadsafe program:

<table>
<thead>
<tr>
<th>Metric</th>
<th>Time</th>
<th>Program</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB2REQCT</td>
<td></td>
<td>14879</td>
</tr>
<tr>
<td>USRCPUT</td>
<td>00:00:01.11961</td>
<td>29763</td>
</tr>
<tr>
<td>SUSPTIME</td>
<td>00:00:01.79190</td>
<td>29763</td>
</tr>
<tr>
<td>DISPWT T</td>
<td>00:00:01.69950</td>
<td>29762</td>
</tr>
<tr>
<td>QRDISPT</td>
<td>00:00:00.37627</td>
<td>14882</td>
</tr>
<tr>
<td>QRCPUT</td>
<td>00:00:00.01568</td>
<td>14882</td>
</tr>
<tr>
<td>KY8DISPT</td>
<td>00:00:03.67361</td>
<td>14880</td>
</tr>
<tr>
<td>KY8CPUT</td>
<td>00:00:01.10212</td>
<td>14880</td>
</tr>
<tr>
<td>L8CPUT</td>
<td>00:00:01.10212</td>
<td>14880</td>
</tr>
<tr>
<td>RMITIME</td>
<td>00:00:03.37489</td>
<td>14880</td>
</tr>
</tbody>
</table>
Non-Threadsafe CICS Exits

DFH$MOLS report of non-threadsafe EXIT:

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value 1</th>
<th>Value 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB2REQCT</td>
<td></td>
<td>14879</td>
</tr>
<tr>
<td>USRCPUT</td>
<td>00:00:01.15467</td>
<td>00:00:02.71036</td>
</tr>
<tr>
<td>SUSPTIME</td>
<td>00:00:02.41534</td>
<td>59518</td>
</tr>
<tr>
<td>DISPWTI</td>
<td>00:00:00.63364</td>
<td>29760</td>
</tr>
<tr>
<td>QRDISPT</td>
<td>00:00:00.01456</td>
<td>29760</td>
</tr>
<tr>
<td>KY8DISPT</td>
<td>00:00:03.35622</td>
<td>29759</td>
</tr>
<tr>
<td>KY8CPUT</td>
<td>00:00:01.14011</td>
<td>29759</td>
</tr>
<tr>
<td>L8CPUT</td>
<td>00:00:01.14011</td>
<td>29759</td>
</tr>
<tr>
<td>RMITIME</td>
<td>00:00:02.92852</td>
<td>14880</td>
</tr>
</tbody>
</table>
Minimizing CPU Overhead

DFH$MOLS of modified program running Threadsafe in test:

EXEC CICS WRITEQ TD replaced with WRITEQ TS

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
<th>Duration</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB2REQCT</td>
<td>00004E20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>USRDISPT</td>
<td>00066339000001E3</td>
<td>00:00:06.69787</td>
<td>483</td>
</tr>
<tr>
<td>USRCPUT</td>
<td>0003A4D30000001E3</td>
<td>00:00:03.82084</td>
<td>483</td>
</tr>
<tr>
<td>SUSPTIME</td>
<td>00025700000001E3</td>
<td>00:00:00.15334</td>
<td>483</td>
</tr>
<tr>
<td>DDISPWTT</td>
<td>000003CE0000001E2</td>
<td>00:00:00.01558</td>
<td>482</td>
</tr>
<tr>
<td>QRDISPT</td>
<td>0000065400000141</td>
<td>00:00:00.02592</td>
<td>321</td>
</tr>
<tr>
<td>QRCPUT</td>
<td>000002B1000000141</td>
<td>00:00:00.01102</td>
<td>321</td>
</tr>
<tr>
<td>KY8DISPT</td>
<td>000659D3000000A1</td>
<td>00:00:06.65937</td>
<td>161</td>
</tr>
<tr>
<td>KY8CPUT</td>
<td>0003A1F7000000A1</td>
<td>00:00:03.80913</td>
<td>161</td>
</tr>
<tr>
<td>L8CPUT</td>
<td>0003A1F7000000A1</td>
<td>00:00:03.80913</td>
<td>161</td>
</tr>
<tr>
<td>QRMODDLY</td>
<td>0000032D000000140</td>
<td>00:00:00.01300</td>
<td>320</td>
</tr>
<tr>
<td>DSCHMDLY</td>
<td>0000033C000000144</td>
<td>00:00:00.01324</td>
<td>324</td>
</tr>
</tbody>
</table>

Copyright (c) 2014, The Evans Group, Inc. & Ivan L. Gelb
Minimizing CPU Overhead - Example

DFH$MOLS of modified program running Threadsafe in test
Results of 10 SQL FETCH placed in Working Storage, then
issue 10 EXEC CICS WRITEQ TD at once

<table>
<thead>
<tr>
<th>Var</th>
<th>Val</th>
<th>Time</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB2REQCT</td>
<td>00004E20</td>
<td>00:00:06.69787</td>
<td>2612</td>
</tr>
<tr>
<td>USRDISPT</td>
<td>000663390000001E3</td>
<td>00:00:03.82084</td>
<td>2612</td>
</tr>
<tr>
<td>USRCPUT</td>
<td>0003A4D30000001E3</td>
<td>00:00:00.15334</td>
<td>2612</td>
</tr>
<tr>
<td>SUSPTIME</td>
<td>000025700000001E3</td>
<td>00:00:00.01558</td>
<td>2611</td>
</tr>
<tr>
<td>DISPWT</td>
<td>000003CE0000001E2</td>
<td>00:00:00.02592</td>
<td>1052</td>
</tr>
<tr>
<td>QRDISPT</td>
<td>00000654000000141</td>
<td>00:00:00.01102</td>
<td>1052</td>
</tr>
<tr>
<td>QRCPUT</td>
<td>000002B1000000141</td>
<td>00:00:00.01102</td>
<td>1052</td>
</tr>
<tr>
<td>KY8DISPT</td>
<td>000659D3000000A1</td>
<td>00:00:06.65937</td>
<td>526</td>
</tr>
<tr>
<td>KY8CPU</td>
<td>0003A1F7000000A1</td>
<td>00:00:03.80913</td>
<td>526</td>
</tr>
<tr>
<td>L8CPU</td>
<td>0003A1F7000000A1</td>
<td>00:00:03.80913</td>
<td>526</td>
</tr>
<tr>
<td>QRMODDLY</td>
<td>0000032D000000140</td>
<td>00:00:00.01300</td>
<td>1050</td>
</tr>
<tr>
<td>DSCHMDLY</td>
<td>0000033C00000144</td>
<td>00:00:00.01324</td>
<td>1055</td>
</tr>
</tbody>
</table>