Engine Hangs, Debugging Methods, and Diagnostic Tools: *A Starters Guide*

Ron Privett (Update of original material created by Jeff Laube)
IBM

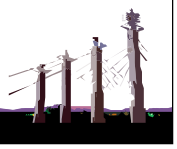
Session D06
Monday, April 28 16:40-17:40

2008 IIUG Informix Conference

www.iiug.org

Introduction

- This presentation aims to help DBA's determine how to deal with critical errors and engine hangs within an IDS environment. It will:
  - Provide guidelines on collecting information after an engine crash
  - Show how to use data collected to isolate and/or reproduce the problem
  - Provide insight into various hangs, and how to accurately define a hang
  - Show examples of identifying the source of a hang, collecting useful diagnostic information and potentially alleviating a hung database



Who is Ron Privett?

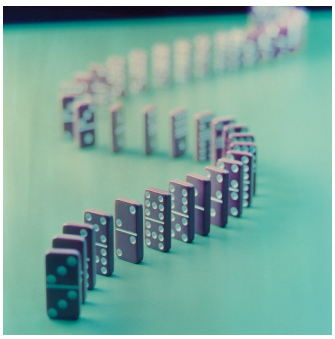
- Officially:
  - Ron is a member of the Advanced Support team, working with the Down Systems and Diagnostic group for IBM Informix engine products. He has worked with IDS for 11+ years, specializing in Replication issues a majority of that time. He has created customer tutorials, developed training materials, trained other engineers how to support ER, and presented at various user groups and user conferences in the past.
- Unofficially:
  - Ron is: a father to 3 boys (13, 9, and 7), the husband to a loving wife, a frustrated Do-It-Yourselfer, a huge U2 fan, and a Soccer fan.

Agenda

- Assertion Failures
- Engine Hangs
- Miscellaneous Diagnostic Methods and Tools

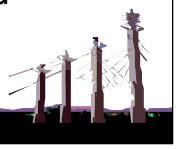
Assertion Failures

- Introduction
- Sample AF files
- Fault Isolation
- Shared Memory Dumps
- Goals in Interpreting AF files



Assertion Failures

- On occasion, an internal problem within an IDS server might present itself as an assertion failure. An assertion failure message is reported in the online.log and details the following:
  - What type of failure has occurred
  - Who is responsible for the failure
  - When the failure occurred
  - Where pertinent information was collected



Sample 1: Assertion Failure Message

```
10:19:03  Assert Failed: No Exception Handler
10:19:03  Informix Dynamic Server Version 9.30.UC1
10:19:03   Who: Session(2, informix@, 0, 0)
               Thread(8, sm_poll, 0, 1)
               File: mtex.c Line: 415
10:19:03   Results: Exception Caught. Type: MT_EX_OS, Context: mem
10:19:03   Action: Please notify Informix Technical Support.
10:19:03  stack trace for pid 16467 written to /tmp/af.3f0af77
10:19:03   See Also: /tmp/af.3f0af77, shmem.3f0af77.0
```

The Default location of the AF file is the /tmp directory, however the location can be changed by an ONCONFIG parameter called DUMPDIR.

In this case - the 'Who done it' is listed on the line:

```
10:19:03   Who: Session(2, informix@, 0, 0)
```

The session id (sid) is **2**, and the username is **informix**. The thread id (tid) is **8**, and the thread name is **sm_poll**, an Internal Thread.

The default location of the ***AF file*** is the ***/tmp*** directory but the location can be changed by an ***ONCONFIG*** parameter which is ***DUMPDIR***.

08:23:18 Who: Session(***2***, ***informix***@, 0, 0)

| |

SID *User Name*

Thread(***8***, ***sm_poll***, 0, 1)

| |

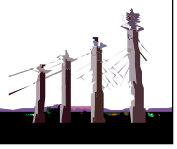
TID *Name of Thread*

The example in the slide shows an internal thread.

Sample 2: Assertion Failure Message

```
09:13:04  Assert Failed: Memory block header corruption detected in
   mt_shm_malloc_segid 7
09:13:04  Informix Dynamic Server Version 9.30.FC2
09:13:04   Who: Session(1599307, idsdb@panaka, 5581, 741958456)
                Thread(1718688, sqlexec, 12c3672e8, 1)
                File: mtshpool.c Line: 3206
09:13:04   Results: Unable to repair pool
09:13:04   Action: Please notify Informix Technical Support.
09:13:04  stack trace for pid 8 written to
   /export/home/informix/log/af.3d885d0f
09:13:04   See Also: /export1/home/informix/log/af.3d885d0f,
   shmem.3d885d0f.0
```

The example in the slide shows a user or SQL thread.

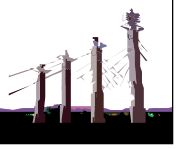


The example in the slide shows a user or SQL thread.

Sample 3: Assertion Failure Message

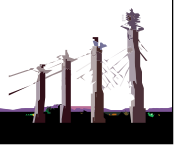
```
08:42:26  Assert Failed:
08:42:26  Informix Dynamic Server Version 7.31.UD4W6
08:42:26   Who: Session(3862, infocel@sumcflo, 24276, 1342723064)
         Thread(3972, srvinfx, 50073c80, 3)
         File: rsnode.c Line: 2255
08:42:26   Results: Delete failed
08:42:26   Action: Run 'oncheck -cI caching_request_3'
```

This shows corruption on a table, and suggests running oncheck to fix it.



Multiple Assertion Failure Messages

- A DBA might notice many consecutive assertion failure messages in the online.log
- Typically, direct primary focus to the initial or 1st reported assertion failure message
- Subsequent assertion failures may be residual effects of the problem first identified by the initial assertion failure message



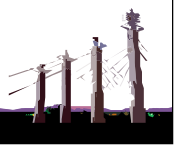
Evidence Collection: evidence.sh

- The onconfig parameter SYSALARMPROGRAM defines the location of the evidence.sh shell script
- Default template is provided at $INFORMIXDIR/etc/evidence.sh
- The script is executed by the server at the point of an assertion failure and is intended to provide insight, primarily through onstat, into the particulars of the engine at the time of failure
- A DBA may choose to add their own components to this file

Evidence Collection: Fault Isolation

- AFWARN - action(s) taken on a warning
- AFFAIL - action(s) taken on a failure
- AFCRASH - action(s) taken on a crash

- Default values are:
  - AFWARN = 0x00000001
  - AFFAIL = 0x00000001
  - AFCRASH = 0x00000201

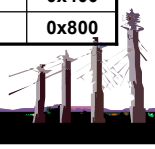


Evidence Collection: Fault Isolation

- Values can be combined to perform multiple actions. For example, 0x203 would place a message in the message log, generate a core dump, and crash the server. See following slide for details on values
- All options will call SYSALARMPROGRAM, unless the flag 0x800 is included
- Another parameter AFLINES will limit the number of Assert Failure messages written to the message log. Setting this to 0 allows all AF messages to be written to the log (the default behavior), and setting it to -1 will not allow any AF messages to be printed

Evidence Collection: Values for AF*

Action/Effect	Value
AFMESSAGE: Display message in the online.log	0x1
AFCORE: Generates a core file	0x2
AFGCORE: Generates a gcore	0x4
AFDUMPSHMEM: Generates a shared memory dump	0x8
AFHANGSERVER: Hang server like AFDEBUG, without respecting BLOCKTIMEOUT	0x10
AFHANGTHREAD: Hang just the thread causing error, not the system	0x20
AFCHECKHANG: Hang thread if no critical resources are held, else hang system	0x40
AFCHECKCRASH: Hang thread if no critical resources are held, else crash system	0x80
AFCHECKRELEASE: Hang thread if no critical resources are held, else return to caller	0x100
AFCRASHSERVER: Crash the server	0x200
AFSERVERSTACK: oninit generates a stack trace	0x400
AFNOEVIDENCE: do not call evidence.sh	0x800

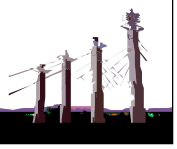


Evidence Collection: Check your settings

- Static Setup
  - Set in onconfig file, then bounce engine
  - Permanently set (meaning a bounce will not reset it)
- Dynamic Setting
  - onparams -m {af config param} {new value}
  - Temporarily set until next bounce
- Viewing Settings
  - onstat -g ras

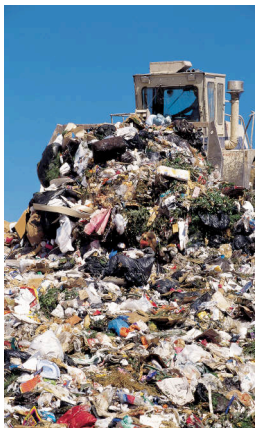
Evidence Collection: AF files

- Output of an assertion failure is written to an Assertion Failure file, or AF file
- The AF file's location is determined by the onconfig parameter DUMPDIR
- Contents of this file include:
  - The original assertion failure message written to the online.log
  - Additional debugging data like hex dumps of associated memory
  - Output of the evidence.sh script



Evidence Collection: Shared Memory Dumps

- A shared memory dump is essentially a file whose contents include all resident, virtual, and communications memory segments used by the server at the point of failure
- Shared memory dumps are written to DUMPDIR
- Enabled by setting the onconfig parameter, DUMPSHMEM to 1, disabled by setting DUMPSHMEM to 0



Ensure DUMPDIR is large enough to hold the complete shared memory dump. You can check the size of Shared Memory using the 'onstat -' command.

Evidence Collection: Shared Memory Dumps

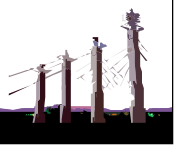
- Shared memory dumps can be an invaluable source of information about an engine failure for a technical support engineer
- Onstat commands can be run against the shared memory dump to gather information that may not have been included in the AF file
- Shared memory dumps allow technical support engineers to examine key control structures of individual threads or for the instance as a whole
- Often, the analysis of a shared memory dump can lead to the identification of a known bug or even uncover a previously unidentified problem in the engine

Evidence Collection: Shared Memory Dumps

To run onstat against a shared memory dump file:

Usage: onstat {infile} {options}

Note: Running onstats against a shared memory file using the {infile} option causes the file to the loaded into memory each time. Depending on the size of the file, and your systems resources - this could take a long time. To avoid this - you could use the 'onstat {infile} -i' so that this file would only be loaded once, run as many commands as needed, then exit



Interpreting the AF Message

- The assertion failure message provides a thread's session id and thread id
- A key component of the AF File (returned by the default evidence.sh file) is the onstat -g sql for the session that failed
- If the AF File does not include the onstat -g sql output for this session and a shared memory dump was taken, you may get it here by running:

  onstat -g sql <session_num> <shmdump_name>

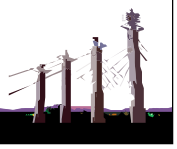
Interpreting the AF Message

The following was pulled from AF File Sample 2:

```
> onstat -g sql 1599307:

Informix Dynamic Server Version 9.30.FC2     -- On-Line -- Up 13 days 16:56:45 --
    1178624 Kbytes

Sess  SQL                    Current             Iso   Lock          SQL  ISAM F.E.
Id    Stmt type              Database            Lvl   Mode          ERR  ERR  Vers
1599307 SELECT         tsmc_prod        CR    Wait 360   0    0    9.03

Current SQL statement :
   SELECT COUNT(*) FROM update_stats  WHERE error_code IS NULL and flag = 2

Last parsed SQL statement :
   SELECT COUNT(*) FROM update_stats  WHERE error_code IS NULL and flag = 2
```

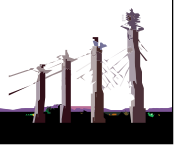
Interpreting the AF Message: Goals

- Why do we want the SQL statement for the failed session?
  - Can you run the statement again, does it reproduce?
  - If failure reproduces, can a test case be created independent of the production environment?
  - A test case is arguably the single most important item you could provide to a technical support engineer
  - A test case
    - can be used to verify relevance to known problems?
    - provides a means of debugging an unidentified issue
  - Identifying an offensive SQL statement is also important in that the statement could be avoided until the problem is addressed



Interpreting the AF Message: Goals

- What if the SQL statement does not reproduce the problem?
  - The default evidence.sh script will attempt to obtain the stack trace of the thread that caused the failure via 'onstat -g stk'
  - It is not uncommon for known bugs to be attributed to a given failure simply by noting similarities in the reported stack traces
  - Shared memory dump analysis by a technical support engineer may be necessary
  - It may also be necessary to incorporate other diagnostic methods (to be discussed later), enabling us to gather more helpful information in the event of a subsequent failure

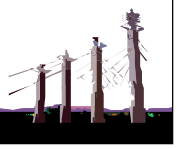


Hangs

- Define the hang
- Network/Connectivity hangs
- Network threads
- Resource hangs
- Evidence Collection

Define the Hang

- Questions to consider…
  - Are only new sessions hung or prevented from connecting to the server?
  - Are all existing sessions still running or are no threads running at all?
  - Is just an individual session hung or not returning after the execution of its last SQL statement?



Network or Connectivity related hangs

- Consider the following:
  - the current protocol (TCP, shared memory, streams)
  - the current INFORMIXSERVER and DBSERVERALIASES settings
  - Is the problem specific to just one protocol? Are shared memory connections fine, but TCP connections hung?
  - Do other server names using the same protocol experience the same problem?

Listen threads and Poll threads

- The Listen thread
  - Responsible for accepting new connections and starting a corresponding session for this new connection
- The Poll thread
  - monitors for incoming events
  - tells the listen thread when new connection requests have arrived
  - notifies existing connections when new messages arrive

Networking thread names

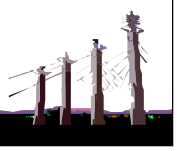
Protocol	Poll Thread	Listen Thread
TLI	tlitcppoll	tlitcplst
Sockets	soctcppoll	soctcplst
Streams	ipcstrpoll	ipcstrlst
Shared Memory	sm_poll	sm_listen

Typical networking thread states

- Consider this onstat -g ath output:

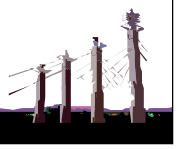
```
Threads:
 tid      tcb      rstcb    prty status            vp-class     name
 9        b338d70  0        2    running             9tli     tlitcppoll
 12       b37a698  0        3    sleeping forever    1cpu     tlitcplst
 13       b308300  0        3    sleeping forever    3cpu     tlitcplst
```

- This output represents typical states for these threads
- Poll threads are generally running
- Listen threads are commonly found sleeping forever
- Poll threads are constantly checking for incoming events. It will wake the listen thread if a new connection indication has arrived
- If a hang has been reported and one or all of these threads are consistently in some other state, this may be an outward indication of the problem



Resource hangs

- When a hang is not related to connectivity, it is most likely waiting on a resource
- The main goal is to isolate the source of the hang
  - single thread waiting on a resource?
  - multiple threads waiting on a common resource?
- A good place to start searching is with the following:
  - onstat -u
  - onstat -g ath

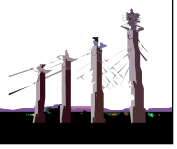


Listing all threads: onstat -u

- Consider the following snip of onstat -u output:

```
Userthreads
address  flags   sessid  user   tty  wait     tout locks nreads nwrites
ae18818  Y--P--- 18      laube  14   b7982b8  0    1     0      0
```

- Examine the wait column which identifies the address of a resource that this thread is currently waiting on
- The wait column address may be associated with items like locks, conditions, buffers or perhaps mutexes
- Examine onstat -k, onstat -g con, onstat -b and onstat -g lmx respectively to check for these items



Find the *wait* resource

- The onstat -g con output corresponds to prior onstat -u

```
Conditions with waiters:
cid      addr     name              waiter   waittime
543      b7982b8  sm_read           40       43
```

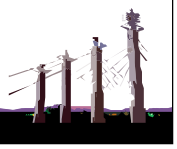
- The thread associated with session 18 is waiting on the sm_read condition. This condition describes the fact that session 18 is associated with a client connected via shared memory and is currently waiting for a new message from the client
- You may need to look for the address of the wait resource within the output of onstat -a or onstat -g all

Listing all threads: onstat -g ath

- onstat -g ath can also be helpful in identifying what type of resource a thread is waiting for:

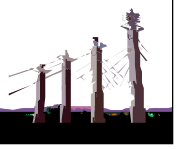
```
Threads:
 tid      tcb      rstcb    prty status              vp-class    name
 40       b436bb0  ae18818  2    cond wait sm_read   1cpu        sqlexec
```

- The thread id associated with session 18 of the previous onstat output is 40
- The relevant portion of the onstat -g ath output shows the status of the thread, and reinforces the output from ‘onstat -g con’, that it is waiting on an sm_read condition



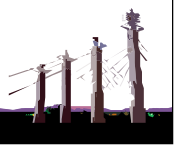
Resource hangs - More to consider

- Are threads waiting on a resource held by another user thread?
- What is the thread doing that holds the resource? Check onstat -g ses output for the corresponding session?
- Is the resource consistently held by the same thread or is it just a 'hot' resource desired and held by many different threads?



Hangs: Evidence collection

- If a hang cannot be remedied by means other than bouncing the server, before you bring the server offline, the following will help technical support diagnose the hang
  - Stack traces of relevant threads (poll threads, listen threads, sqlexec threads, etc.)
  - onstat -a
  - onstat -g all
  - A shared memory dump

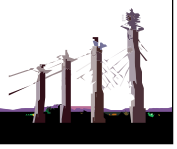


Obtaining a stack trace

- Consider the following onstat -g ath output:

```
Threads:
 tid      tcb      rstcb    prty status            vp-class      name
 9        b338d70  0        2    running             9tli      tlitcppoll
 12       b37a698  0        3    sleeping forever    1cpu      tlitcplst
 13       b308300  0        3    sleeping forever    3cpu      tlitcplst
```

- When a thread does not have a status of running, the best method to obtaining a stack trace is with onstat -g stk <thead_id>
- The command, onstat -g stk 12, will return the stack trace for the listen thread having a thread id of 12



Stack traces of running threads

- You can use onmode -X stack <vpid>, to generate a stack of the thread running on the VP listed
- For example: `onmode -X stack 9`, will generate a stack trace for the thread running on VP 9
- The onstat -g ath output from previous slides confirms this to be the tli NET VP that the tlitcppoll thread is running on
- A message like the following in the online.log will report the location of the file that contains the stack trace

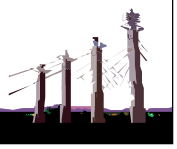
```
11:38:33  stack trace for pid 17327 written to /tmp/af.3f11399
```

Stack trace of running threads

- In older IDS versions, where onmode -X is not available (7.x and pre-9.21)
- The command, kill -7 <vpid>, generates a stack trace for the thread running on the virtual process with id, vpid.
- The tlitcppoll thread from prior onstat -g ath output is running on vp-class, 9tli.
- Examine onstat -g sch to associate 9tli with a process id

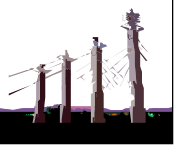
```
VP Scheduler Statistics:
 vp     pid       class       semops     busy waits  spins/wait
 9      17327     tli         2          2           1000
```

- The process id for VP 9 is 17327. The command to obtain the stack trace of the running tlitcppoll thread is:
- kill -7 17327
- A message will be written to the online.log describing location of stack



Hung systems: Shared memory dumps

- Shared memory dumps may also provide great insight for technical support into hung systems
- They are not automatically generated as with assertion failures
- The command, onstat -o <filename>, can be executed to manually obtain a shared memory dump



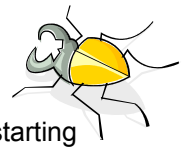
Summary: Engine hangs

- Diagnosing engine hangs is not an exact science
- Different hangs warrant different diagnostic methods
- This section has described general considerations for determining what type of hang you may be encountering and information that can be gathered to help diagnose the hang

Misc. Diagnostic Methods and Tools

- AFDEBUG
- SQLIDEBUG
- Error Trapping
- CCFLAGS
- xtrace
- Client IP Trace

AFDEBUG

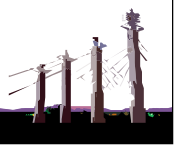


- Turning on
  - set AFDEBUG environment variable to 1 before starting engine
  - onmode -A 1
- Turning off
  - onmode -A 0
- Description
  - In the event of an assertion failure, if AFDEBUG is turned on, the engine will suspend the process that failed. The assertion failure message in the online.log will indicate that you can attach to the corresponding process id with a debugger

- May be useful if having trouble obtaining stack trace. Its use has lessened as shared memory dump analysis skills have heightened, and product quality has increased

SQLIDEBUG

- Messages are sent between a client and the IDS via an internal protocol, SQLI
- Depending on the issues of a particular case, a technical support engineer may request the capture of these messages to understand what transpired up to a particular event
- Typically, SQLI messages are captured at the client using the environment variable SQLIDEBUG
  - Setting SQLIDEBUG to '2:/tmp/sqli.out' before running the client will cause the application to write all SQLI messages to the file /tmp/sqli.out appended with the process id
- Starting with version 10, SQLIDEBUG has been implemented so that SQLI messages can also be captured from the server
- Technical support has a utility to interpret the binary file generated by SQLIDEBUG



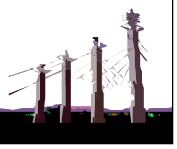
Note: A full session on SQLIDEBUG is scheduled for Wednesday at 1PM

Error trapping with IDS

- This command can be used to generate an assertion failure if the error, error_num, should occur within the server
  - onmode -I <error_num>[,<session_id>]
- For example, the command below will instruct the engine to generate an assertion failure if a syntax error is encountered
  - onmode -I 201
- This command clears all error trapping in the server
  - onmode -I
- You can use this in conjunction with AFDEBUG to hang the server if a particular error is encountered
- The following command will release the server when it is hung on a trapped error
  - onmode -i

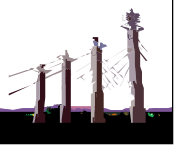
CCFLAGS

- CCFLAGS is used by technical support to turn on various consistency checking within the IDS. This capability allows the engine to report a problem, ideally, well before an assertion failure would occur, thus providing a more accurate account of the details surrounding an engine failure
- Examples of turning on:
  - onmode -f 0x400
- set CCFLAGS environment variable to 0x400 before starting engine
- Example of turning off:
  - onmode -f 0x0



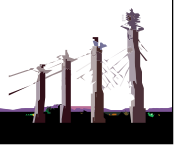
CCFLAGS Examples

- *Memory pool checking on memory allocation and frees.* Used when memory is being corrupted by another process/thread
- *Memory scribble.* Used to place a pattern in memory when allocated or deallocated
- *Checking global memory each second.* Used to see memory leaks
- *Btree consistency checking.* Used to check index health



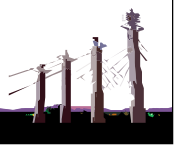
XTRACE

- Tracing of various engine components exists within the server and may be turned on dynamically with the IBM/Informix utility, xtrace, that is located in $INFORMIXDIR/bin
- The xtrace buffer is circular, and will overwrite the oldest information
- The following example sets the xtrace buffer size to accommodate 1000 trace events. Various components can be traced. The example below illustrates how to trace the server's TLI networking component
  - xtrace size 1000
  - xtrace heavy -c XTF_TLI -f XTF_SYSCALLS
  - xtrace on
  - xtrace fview > xtrace.out



Client IP Trace

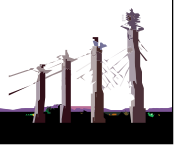
- Ever see a -27001 error message repeatedly in the online.log? Who is trying to connect and failing?
- In a scenario where multiple clients (10's, 100's or sometimes 1000's) request a connection, it can be difficult to diagnose and find the source. In such a situation, it is necessary for the DBA to know the client connection details, so that further action could be taken to rectify the source of the issue



Client IP Trace (cont)

Procedure to trace the client IP

1. Check the status of the IDS server. Must be on-line
2. Set and check the tracing variable values (optional)
   - xtrace size 8000 (default 4096)
   - xtrace info
3. Initialize the IP trace components:
   Component=XTF_IPTRACE, Function=XTF_SYSCALLS
   - xtrace heavy –c XTF_IPTRACE –f XTF_SYSCALLS



Client IP Trace (cont)

4. Start the tracing: `xtrace on`
5. View the client connection details:
   `xtrace view`

Xtrace messages:

For Sockets:

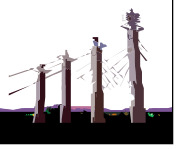
```
accpsocket - Accepted IP Address <Hex Address> [<Host Name>]
accpsocket - Rejected IP Address <Hex Address>[<Host Name>]
```

For TLI:

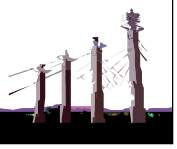
```
accptli - Rejected IP Address <Hex Address> [(<Host Name>)]
accptli - Queuing IP Address <Hex Address> [(<Host Name>)]
accptli - Failed accept IP Address <Hex Address> [(<Host Name>)]
accptli - Accepting IP Address <Hex Address> [(<Host Name>)]
```

*** Host Name not printed when host name is not known ***

6. End the tracing session: `xtrace off`



Any comments or questions ?



Contact Info:

- Session = D06
- Title = Engine Hangs, Debugging Methods, and Diagnostic Tools: *A Starters Guide*
- Speaker = Ron Privett
- Company = IBM
- e-mail = rprivett@us.ibm.com