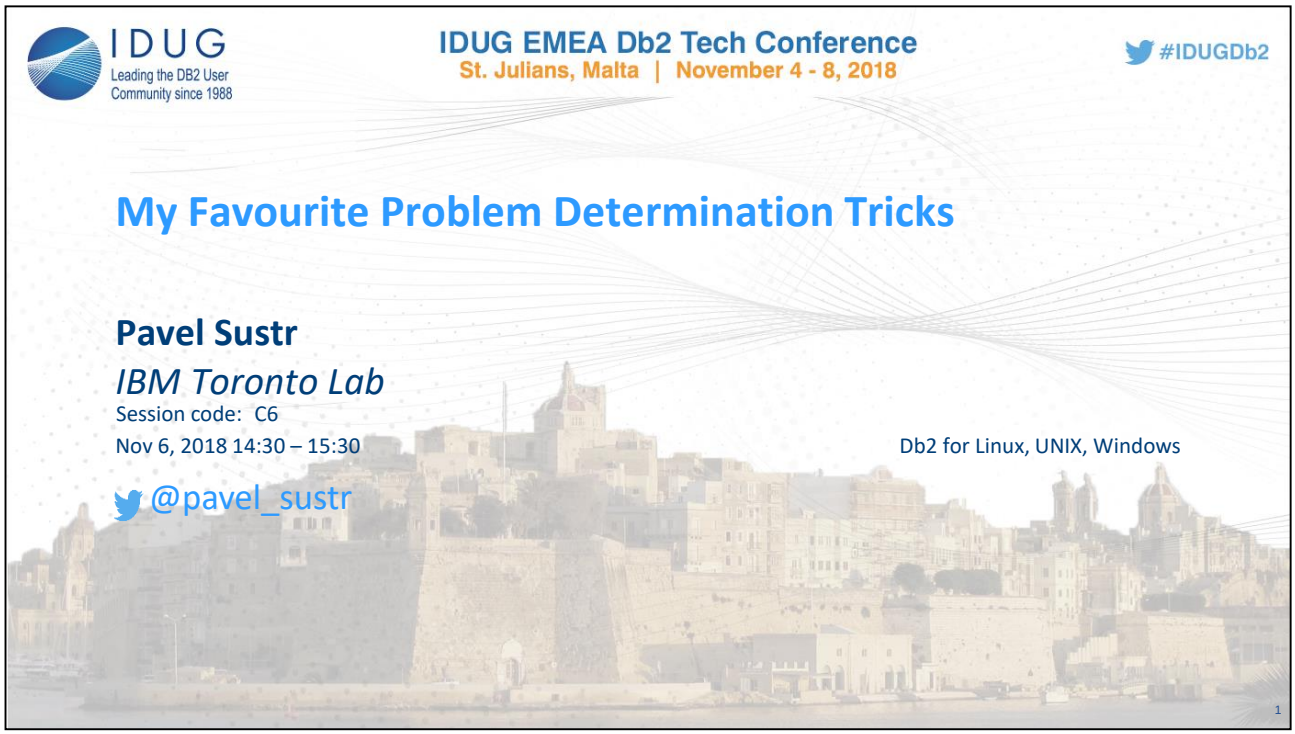


Pavel's Bio

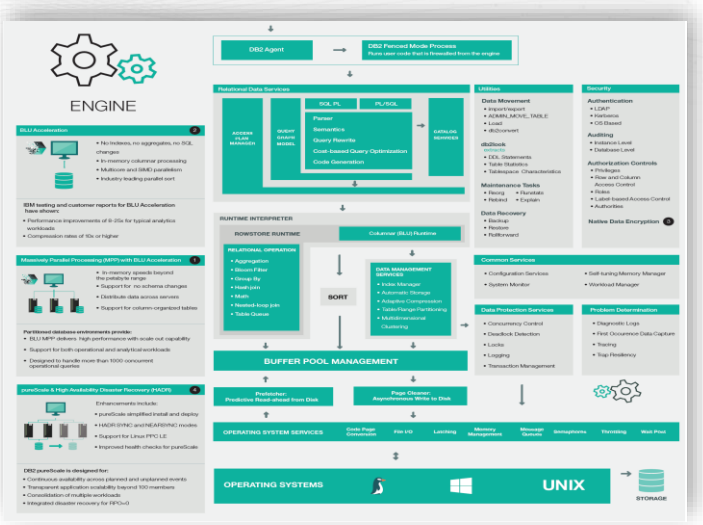
Senior Manager and Senior Software Engineer with IBM Db2 LUW development, responsible for multiple core Db2 kernel components. Always thrilled to work on hard-to-crack puzzles. Expertise in Db2 LUW kernel architecture, configuration and administration, advanced problem determination, memory architecture, memory leak troubleshooting, and assembly language. Hands-on development experience with buffer pool management, storage, prefetching, page cleaning, transaction logging, recovery, monitoring, and problem determination. As a member of the Db2 team, Pavel spent years in Db2 L2/L3 advanced support (over 1,500 resolved cases), then transitioned to Db2 LUW kernel development. In his past life Pavel was an application developer mostly using C++, SQL, .NET, Oracle, MS-SQL, and Informix on Windows, Linux, Solaris, and HP-UX.

Agenda

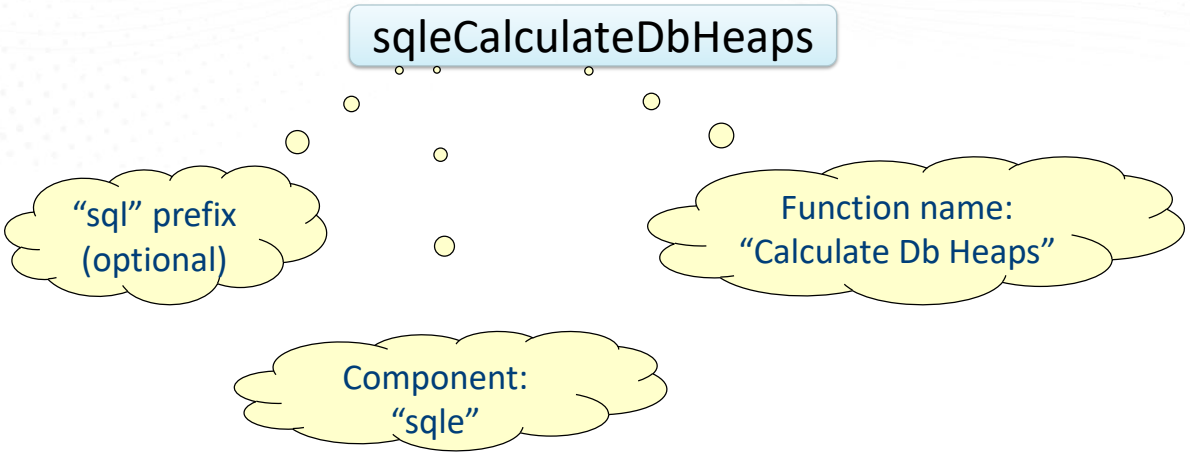
- Db2 Kernel and Return Code Anatomy
- Fifty shades of Db2 Trace
- Call-Out Scripts
- Locking/Latching Tricks
- Table Space Map
- Sleep... ☺

Db2 Engine Anatomy

- Millions of lines of code
- Grouped into components
- Component expertise
  - Single : hard
  - Multiple : lifetime effort
  - Most : Keri ☺



Routine Naming Anatomy



4

- The "sql" prefix. Usually associated with the most mature Db2 components. The "sql" part is not neccesarily referring to the SQL language. Instead, it has a broader usage scope, as in "SQL engine" = "Db2 Kernel". Newer components (e.g. CDE) do not use the prefix.
- Component. When the "sql" prefix is present, the prefix becomes part of the component's name. E.g. "sqle" = "Process Model", "sqlb" = "Buffer Pool Services", etc… For a list of frequent components, see the reference slides at the end of the presentation.
- User (developer) friendly name. Can often be used to guess the purpose of the routine.

Function names are often a good string to search for when it comes to APARs, diagnostic log entries, Db2 trace, etc…

Making Sense of Return Codes

- *Wikipedia: In computer programming, a **return code** or an **error code** is an enumerated message that corresponds to the status of a specific software application*
- Db2 return codes can be spotted in various places: Db2 diagnostic log, notification log, traces, command line, CLI logs, JAVA traces
- Most of the time they should be accompanied by a text message
  - What if this is not the case?
  - What if I want to know more?

db2diag –rc/db2diag -cfrc

- The `db2diag` tool can be used to translate a return code to the corresponding text representation
  - `db2diag -rc <code>` for non-pureScale return codes
  - `db2diag -cfrc <code>` for CF return codes
- The `<code>` can be any of the following:
  - Hex code, e.g.: `0x870F0016`
  - Decimal code, e.g.: `-2029060074`
  - Mnemonic name, e.g.: `SQLO_SHAR`

Example: db2diag –rc

```
$ db2diag -rc 0x870F0016

ZRC class :
        Global Processing Error (Class Index: 7)
Component:
        SQLO ; oper system services (Component Index: 15)
Reason Code:
        22 (0x0016)

Identifer:
        SQLO_SHAR
Identifer (without component):
        SQLZ_RC_SHAR

Description:
        File sharing violation.
```

7

The output is a bit shortened for easier viewing. The entire message looks like:

> db2diag -rc SQLO_SHAR

Input ZRC string 'SQLO_SHAR' parsed as 0x870F0016 (-2029060074).

ZRC value to map: 0x870F0016 (-2029060074)
    V7 Equivalent ZRC value: 0xFFFFF616 (-2538)

ZRC class :
    Global Processing Error (Class Index: 7)
Component:
    SQLO ; oper system services (Component Index: 15)
Reason Code:
    22 (0x0016)

Identifer:
    SQLO_SHAR
Identifer (without component):
    SQLZ_RC_SHAR

Description:
    File sharing violation.

Associated information:

Sqlcode -902

SQL0902C A system error occurred. Subsequent SQL statements cannot be processed. IBM software support reason code: "".

Number of sqlca tokens : 1

Diaglog message number: 8519

Return Code Anatomy

- ZRC class :

  Non-Critical Media Error (Class Index: 4)

- Component:

  SQLO ; oper system services (Component Index: 15)

- Reason Code:

  1 (0x0001)

8

- Return codes are often remapped when the code is leaving a component's scope and entering another component's scope (for explanation on components, see subsequent slides). For example: SQLO_ACCD => SQLZ_RC_ACCD => SQL0970N The system attempted to write to a read-only file.
- The lowest level error consists of:
  1) Class
  2) Component
  3) Reason code

Fifty Shades of Db2 Trace

- Db2 trace typically provides information on:
  - Internal functional calls made
  - Code path used, i.e. *code flow*
  - Data being manipulated at each point within the function
  - Time elapsed in each function, if enabled

However, Db2 trace can also be used to perform other neat tricks!

•Db2 traces are invoked by issuing the **db2trc** command from an operating system command prompt.

•When invoked, trace points within the Db2 source will 'fire' during runtime.

•The firing of each trace point causes information such as the location within the code, error codes, return codes, and certain variables to be written to a buffer.

•**db2trc** allows for administration of the facility and parsing and formatting of the trace dump files.

Trace: Sample Usage

- Typical Db2 trace invocation (*"trace everything"*):

```
db2trc on -l <buffer_size> -t
```

- Trace specific components:

```
db2trc on -l <buffer_size> -t -Madd SQLB -Madd SQLD
```

- Trace specific applications:

```
db2trc on -l <buffer_size> -apphdl <apphdl>          (up to 16 apphandles), OR
db2trc on -l <buffer_size> -appid <applid>           (up to 12 application IDs)
```

- Trace into a file (*"unlimited"* buffer):

```
db2trc on -f <dmpfile> -t
```

- Verify trace is on:

```
db2trc inf
```

- Dump the trace buffer, turn off tracing, and format trace data:

```
<recreate the problem>
db2trc dmp <dmpfile>                                 (skip if tracing into a file, -f, is used)
db2trc off
db2trc flw <dmpfile> <flwfile>
db2trc fmt <dmpfile> <fmtfile>
```

-l [*buffersize*]

- This option specifies the size and behavior of the trace buffer. -l specifies that the last trace records are retained (that is, the first records are overwritten when the buffer is full). The buffer size can be specified in either bytes or megabytes. To specify the buffer size in megabytes, add the character M | m to the buffer size. For example, to start db2trc with a 4–megabyte buffer: db2trc on -l 4m The default and maximum trace buffer sizes vary by platform. The minimum buffer size is 1 MB. The buffer size must be a power of 2.

[-t]

Include timestamps.

Trace: Flow

```
308986      sqleProcessSCoordRequest entry [eduid 37 eduname db2agent]
310069      | sqlpParallelRecovery entry [eduid 37 eduname db2agent]
              <...lots of other calls here...>
316955      | sqlpParallelRecovery exit [rc = SQLB_EMP_MAP_INFO_NOT_FOUND]
317046      sqleProcessSCoordRequest exit
```

- **Unique trace ID. Increasing order, trace always starts with 1.**
- **Db2 function called. Name chosen by Db2 developers, often self-explanatory.**
- **Specific place in function. Could be "entry", "exit", "probe number", "marker", …**
- **Db2 "thread" (EDU) ID and name. Matches the EDU ID and name in db2diag.log.**
- **Return code. A good string to search for in Db2 APARs.**

FLW provides a visual representation of which Db2 routines were called and by whom, their return code, markers, and probe points. The trace IDs are not sequential (i.e. contain "holes") because of context switching, i.e. EDU "A" may own entries 1 and 3, but EDU "B" running in parallel will own 2 and 4.

EDU is a Db2 term for "thread". Stands for "Engine Dispatchable Unit".

Trace: Format

```
316955    exit DB2 UDB recovery manager sqlpParallelRecovery fnc (2.3.94.48.0) pid 14925 tid
   46912874998080 cpid 14546 node 0 rc = 0x8402001B =
          -2080243685 = SQLB_EMP_MAP_INFO_NOT_FOUND

316956    entry DB2 UDB base sys utilities sqleSubCoordTerm fnc (1.3.5.1051.0) pid 14925 tid
   46912874998080 cpid 14546 node 0 eduid 37 eduname
          db2agent
```

- **Unique trace ID. Matches the ID in FLW.**
- **Specific place in function. Could be "entry", "exit", "probe number", "marker", …**
- **Db2 area, component, and function called. Note the unique "IP address".**
- **Process/thread/EDU/Node ID, EDU name. Also could contain timestamp, etc…**
- **Return code. Same as in FLW.**

FMT provides additional detail about individual trace entries. Unlike FLW, the entries are perfectly sequential and ordered by time. When timestamps are present (db2trc –t), these entries could be used for performance measurements. Because of the aforementioned context switching, extra attention needs to be paid to EDU which owns the trace entry of interest.

Trace: Flow and Timeline

```
$ db2trc flw -t trace.dmp trace.flw
```

Trace ID

Code flow

Time

```
34792          13.644484000   | | | sqlbDMSGetOpenInfo entry [eduid 26 eduname db2pfchr]
34795          13.644485000   | | | | SqlbFhdlTbl::getFileHandle entry [eduid 26 eduname db2pfchr]
34797          13.644485000   | | | | | SqlbFhdlTbl::getHashOpts entry [eduid 26 eduname db2pfchr]
34801          13.644486000   | | | | | SqlbFhdlTbl::getHashOpts exit
34803          13.644487000   | | | | | SqlbFhdlTbl::findSlot entry [eduid 26 eduname db2pfchr]
34806          13.644487000   | | | | | SqlbFhdlTbl::findSlot exit
34809          13.644488000   | | | | SqlbFhdlTbl::getFileHandle exit
34811          13.644488000   | | | sqlbDMSGetOpenInfo exit
34813          13.644489000   | | | sqloReadV entry [eduid 26 eduname db2pfchr]
34816          13.644489000   | | | | sqloReadVLow entry [eduid 26 eduname db2pfchr]
37638          13.645357000   | | | | sqloReadVLow exit
37642          13.645359000   | | | sqloReadV exit
```

- Time spent per EDU
- Look for *"lags"* in the time sequence

13

For example:

```
34816          13.644489000   | | | | sqloReadVLow entry [eduid 26 eduname db2pfchr]
37638          13.645357000   | | | | sqloReadVLow exit
```

This routine performs a disk read of one page. The difference between the two entries is 0.000868, meaning reading one page took 0.000868 s.

Trace: Performance Trace

```
$ db2trc on -perfcount -t
<…run your scenario…>
$ db2trc dmp trace.dmp
$ db2trc off
$ db2trc perffmt trace.dmp trace.perfmt
$ sort -k2nr trace.perfmt > trace.perfmt.sorted
```

Time spent (s)

Number of executions

```
             1        15.725198000 sqlrr_execimmd
             1        15.725046000 sqlrr_execute_immed
             1        15.702721000 sqlriSectInvoke
        524288        11.086911000 sqlrinsr
        524288        10.367470000 sqldRowInsert
        262145         8.946307000 sqlriisr
```

Routine names

- A great way to “profile” what is happening in Db2

Trace: analyzetrace (COMING SOON ™)

- Previous example shows a "*per-instance*" performance profile
- What if you want to see "*per-EDU*" performance data?

```
$ db2trc on -f trace.dmp -t
$ db2 connect to sample
$ db2trc off
$ db2trc flw -t trace.dmp trace.flw
$ ./analyzetrace -f trace.flw
Output will be sorted by total time in descending order

Slurping file trace.flw ..

Slurping 223909 lines in trace.flw
Sorting .. please wait
Please check perftrace.out
Program Finished
```

15

- Credit to Rajib Sarkar
- The tool uses a formatted Db2 trace file, either FLW or FMT, as the input
- The formatted Db2 trace file must contain timestamps (db2trc should have been invoked using the –t option)
- Note that "analyzetrace" is currently not shipped, we are working on shipping this soon
- Once done, the tool will be located in the usual ~/sqllib/pd directory
- In the meantime, contact me for the current version

Trace: analyzetrace Example

```
Pid   Lvl FuncName                          TTime(ms) HTime    LTime    AvgTime  NCalls ERHTime
----------- --- ------------------------- -------- -------- -------- -------- ------ -------
21160(Tid = 139923394914048, Node = 0)
        7 sqljcReceive                    2194.898 1138.186 1056.712 1097.449  2  222718
        7 sqljsParse                       851.295  851.295  851.295  851.295  1   24479
        8 sqljsParseConnect                851.263  851.263  851.263  851.263  1   24509
       10 sqljsConnectAttach               851.250  851.250  851.250  851.250  1   24513
       11 sqleUCagentConnect               851.211  851.211  851.211  851.211  1   24516
       12 sqleUCengnInit                   851.183  851.183  851.183  851.183  1   24541
       13 sqeApplication::AppLocalStart    851.178  851.178  851.178  851.178  1   24542
       15 sqeApplication::AppStartUsing    850.534  850.534  850.534  850.534  1   25039
       17 sqeLocalDatabase::FirstConnect   490.520  490.520  490.520  490.520  1   25623
       18 sqledint                         446.490  446.490  446.490  446.490  1   30916
       19 sqlbinit                         243.776  243.776  243.776  243.776  1   40509
       19 sqlpinit                         186.967  186.967  186.967  186.967  1   31061
       20 sqlpgint                         154.109  154.109  154.109  154.109  1   37125
       20 sqlbInitBufferPool                91.588   22.036   14.973   18.318  5   44170
       21 sqlbSetupClnrGroupForBP           86.403   21.557   14.391   17.281  5   44287
```

16

- Profile of the CONNECT thread
- All times in miliseconds
  - Pid > Process id
  - Lvl > Depth at which function found (counting the pipe signs in the flw output)
  - FuncName > Function Name
  - TTime > Total Time spent in the function
  - HTime > Highest Time spent by 1 call in this function
  - LTime > Least Time spent by 1 call in this function
  - AvgTime > Avg. Time spent by 1 call in this function
  - ERecnumHtTime > Entry Record number for the highest time call to the function
- We can clearly see how much time it took to initialize individual Db2 comments: sqlbinit, sqleinit, sqlpinit, …
- How many buffer pools are there? How long did it take to allocate them?

Trace: Print Call Stack

```
$ db2trc print -stack 314032 trace.flw

pid = 14925 tid = 46912874998080 node = 0

308986 sqleProcessSCoordRequest entry [eduid 37 eduname db2agent]
310069 | sqlpParallelRecovery entry [eduid 37 eduname db2agent]
314023 | | sqlpPRecReadLog data [probe 1250]
314027 | | | sqlprProcDPSrec data [probe 430]
314028 | | | | sqlpRecDbRedo entry [eduid 37 eduname db2agent]
314030 | | | | | sqldmrdo data [probe 0]
314031 | | | | | | sqldomRedo entry [eduid 37 eduname db2agent]
314032 | | | | | | | sqldRedoFastTruncTable entry [eduid 37 eduname db2agent]
```

- If you only consider the initial entry for each routine in a Db2 trace flow file, you will get a *"call stack"* – an ordered sequence of internal Db2 calls.

Trace: Suspend Db2

```
$ db2trc on -debug "DB2.SQLB.sqlbinit.entry" -suspend
$ db2 connect to sample
<…hangs…>

db2diag.log
2018-10-15-12.50.06.307166-240 I135142E2673          LEVEL: Severe
PID     : 10253                TID : 140494935942912 PROC : db2sysc 0
INSTANCE: db2inst2             NODE : 000            DB   : SAMPLE
APPHDL  : 0-79                 APPID: *LOCAL.db2inst2.181015165006
AUTHID  : DB2INST2             HOSTNAME: demobox
EDUID   : 18                   EDUNAME: db2agent (SAMPLE) 0
FUNCTION: DB2 UDB, trace services, crash_trace, probe:10
MESSAGE : MARKER=16397=PD_DB2_TRC_CRASH_SUSPEND "Trc. debug: Suspending"
DATA #1 : Function, 4 bytes
DB2 UDB, buffer pool services, sqlbinit
```

- Suspend Db2 during an event of your choice
- Can also be used to crash Db2 (great for recovery tests ☺)

- The **-debug** flag is followed by a place identifier, use **db2trc on –u** to display help
- Alternatives to **–suspend**:
  - **-crash** – crashes the EDU
  - **-sleep <n>** – pauses the EDU for <n> seconds
- Use **db2trc chg -resume -debug "DB2.SQLB.sqlbinit.exit" –suspend** to move the suspension point to the exit of the same routine
- Use **db2trc off** to disable the suspend

Trace: Call-Out Script

- *Situation: You want to collect information when a specific Db2 routine is executed.*
- Solution:
  - `db2trc -debug -db2cos`
- This action triggers the `db2cos` script located in `~/sqllib/bin`
- If you want to customize the script:
  1. Copy `~/sqllib/bin/db2cos` to `~/sqllib/adm`
  2. Locate the "`DB2_TRC`" section
  3. Add your commands

- An example of how to customize the script:

```
"DB2_TRC")
      echo "Trace Point Caught"                                  >> $logfile
      db2trc dump /tmp/trc.dmp           >> $logfile
      db2trc off                                        >> $logfile
      echo "Instance         " $instance               >> $logfile
      echo "Database:        " $database               >> $logfile
      echo "Partition Number:" $dbpart                 >> $logfile
      echo "PID:             " $pid                              >> $logfile
      echo "TID:             " $tid                              >> $logfile

<…add your own commands…>
```

Trace: Call-Out Script Example

```
$ db2trc on -debug "DB2.SQLB.sqlbinit.entry" -suspend -db2cos
$ db2 connect to sample
<…hangs…>

db2diag.log
2018-10-15-14.20.53.747332-240 I6776E379            LEVEL: Event
PID     : 24321                TID : 140385756596000 PROC : db2vend (PD Vendor Process - 18)
INSTANCE: db2inst2             NODE : 000
HOSTNAME: demobox
FUNCTION: DB2 UDB, trace services, pdInvokeCalloutScriptDirect, probe:10
START   : Invoking /home/db2inst2/sqllib/adm/db2cos from buffer pool services sqlbinit

24178.18.000.cos.txt
Trace Point Caught
Instance          db2inst2
Database:         SAMPLE
Partition Number: 000
PID:              24178
TID:              3179276032
```

20

You can conclude this technique by dumping the trace leading to this point, e.g,:

- change your directory to **$HOME/sqllib/db2dump/**
- **db2trc dmp trace.dmp** (this will dump the trace buffer into a file)
- **db2trc off** (this will stop tracing and DB2 will resume)
- **db2trc fmt trace.dmp trace.fmt**
- **db2trc flw trace.dmp trace.flw**
- **db2support . -d <dbname> -c -g -s**

db2pdcfg: Execute Call-Out Script

- `db2pdcfg` can also be used to execute the call-out script
  - `db2pdcfg -catch diagstr="Message to capture"`
  - The string must be located in the "MESSAGE" section of the diagnostic log entry
  - Use of a substring is acceptable

db2trc -debug -db2cos	db2pdcfg -catch diagstr
Fires off when a routine/probe is executed	Fires off when a diagnostic message is encountered
Use when a diagnostic message is not present	Use when unsure about the routine name
Use when one routine produces multiple messages	Use when multiple routines produce the same message

db2pdcfg: Call-Out Example (1)

For example, let us capture the following event which happens during the database activation time (e.g. during the first connection):

```
2018-10-02-16.02.57.310281-240 I3547E521             LEVEL: Event
PID     : 20436                TID : 140319605647104 PROC : db2sysc 0
INSTANCE: db2inst2             NODE : 000            DB   : SAMPLE
APPHDL  : 0-18                 APPID: *LOCAL.db2inst2.181002200256
AUTHID  : DB2INST2             HOSTNAME: demobox
EDUID   : 18                   EDUNAME: db2agent (SAMPLE) 0
FUNCTION: DB2 UDB, catcache support, sqlrlc_catcache_init, probe:260
MESSAGE : Catalog cache size:
DATA #1 : unsigned integer, 8 bytes
851968
```

- Let us pick “Catalog cache size” without the trailing colon

db2pdcfg: Call-Out Example (2)

```
$ db2pdcfg -catch diagstr="Catalog cache size"
<...skipping...>
   Action:         Error code catch flag enabled
   Action:         Execute /home/db2inst2/sqllib/bin/db2cos callout script
   Action:         Produce stack trace in db2diag.log

$ db2 connect to sample

db2diag.log
FUNCTION: DB2 UDB, RAS/PD component, pdLogInternal, probe:999
DATA #1 : <preformatted>
Caught String Catalog cache size. Dumping stack trace

2018-10-02-16.02.57.378838-240 I6624E380            LEVEL: Event
PID     : 20633                TID : 140296236762912 PROC : db2vend (PD Vendor Process - 18)
INSTANCE: db2inst2             NODE : 000
HOSTNAME: demobox
FUNCTION: DB2 UDB, trace services, pdInvokeCalloutScriptDirect, probe:10
START   : Invoking /home/db2inst2/sqllib/bin/db2cos from RAS/PD component pdLogInternal
```

- When satisfied, use "db2pdcfg –catch –clear" to clear the catch flag settings

One Minute Lock Problem Determination

- ***Situation:*** *A connection to the database hangs or the database is slower than usual, and you want to investigate possible lock contentions*
- Solution:
  - `db2pd -db <dbname> -locks -wlocks`
- This command will give you:
  1. Information on all locks currently used by the database (including those that nobody is waiting for)
  2. Holder/Waiter info for transaction locks being waited on

Lock Example (1)

```
SESSION 1
$ db2 connect to sample

$ db2trc on -debug "DB2.SQLB.sqlbDMSMapAndRead.entry" -suspend
Trace is turned on

$ db2 "select count(*) from staff"
<...hangs...>

SESSION 2
$ db2 connect to sample

$ db2 list tables
<...hangs...>
```

25

- We are using db2trc to simulate a hang of two independent EDUs

Lock Example (2)

```
$ db2pd -db sample -locks -wlocks
Database Member 1001 -- Database SAMPLE -- Active -- Up 0 days 00:28:21 -- Date 2018-10-22-12.05.40.964806

Locks:
Address            TranHdl Lockname                  Type       Mode Sts Owner Dur HoldCount  Att        ReleaseFlg
0x00007F66DC2F2E00 13      0100000001000000100C07ED6 VarLock    ..S  G   13    1   0          0x00000000 0x40000000
0x00007F66DC2F5F00 12      434F4E544F4B4E3128DD6306C1 PlanLock   ..S  G   3     1   0          0x00000000 0x40000000
0x00007F66DC2E7980 3       41414141416641647CF81EA4C1 PlanLock   ..S  G   3     1   0          0x00000000 0x40000000
0x00007F66DC2F5E80 12      41414141416641647CF81EA4C1 PlanLock   ..S  G   3     1   0          0x00000000 0x40000000
0x00007F66DC2F2D00 13      5359534C564C3031DDECEF28C1 PlanLock   ..S  G   13    1   0          0x00000000 0x40000000
0x00007F66DC2E7B80 3       E08172E4667F000000000001C1 PlanLock   ..X  G   3     1   0          0x00000000 0x40000000
0x00007F66DC2F5E00 12      E08172E4667F000000000001C1 PlanLock   ..X  W   3     0   0          0x00000000 0x00000000
0x00007F66DC2F2F00 13      0500040000000000000000054 TableLock  .IS  G   13    1   1          0x00002000 0x40000000
0x00007F66DC2E7D80 3       00000D0000000000000000054 TableLock  .IS  G   3     1   0          0x00002000 0x40000000
0x00007F66DC2F1280 14      00000D0000000000000000054 TableLock  .IN  G   14    1   0          0x00003000 0x40000000
0x00007F66DC2F2A80 7       0000070000000000000000054 TableLock  .IX  G   7     1   0          0x00202000 0x40000000

Database Member 1001 -- Database SAMPLE -- Active -- Up 0 days 00:28:21 -- Date 2018-10-22-12.05.40.973305
Locks being waited on :
AppHandl [nod-index] TranHdl Lockname                  Type       Mode Conv Sts CoorEDU AppName
8        [000-00008] 3       E08172E4667F000000000001C1 PlanLock   ..X       G   18      db2bp
32       [000-00032] 12      E08172E4667F000000000001C1 PlanLock   ..X       W   46      db2bp
```


- In the "-locks" output, look for "W" (waiting) and the corresponding "G" (granted)
- Or, simply have a look at the "-wlocks" output which will sort this out for you
- In this case, application handle 8 is holding a plan lock in X, and application handle 32 is waiting for this lock
- You can use MON_FORMAT_LOCK_NAME to obtain extended information about the lock:

```
$ db2 "SELECT SUBSTR(NAME,1,20) AS NAME, SUBSTR(VALUE,1,50) AS VALUE FROM TABLE(
MON_FORMAT_LOCK_NAME('E08172E4667F000000000001C1')) as LOCK"

NAME                 VALUE
-------------------- --------------------------------------------------
LOCK_OBJECT_TYPE     PLAN
PACKAGE_TOKEN        ŕrä
INTERNAL             HashPkgID:01000000,LoadingBit:1

  3 record(s) selected.
```

One Minute Latch Problem Determination

- ***Situation:*** *Similar to the locking case, except there are no lock holders or waiters present*
- Solution:
  1. `db2pd -latches`
     - Provides a quick overview of latch holders/waiters
     - Use when you do not care about the root cause
  2. `db2pd -stack all` followed by `analyzestack -l`, OR
     `db2fodc -hang basic` followed by `analyzestack -l`
     - More details, perhaps harder to read initially
     - Often sufficient for root cause analysis

Latch Scenario

```
SESSION 1
$ db2 connect to sample

$ db2trc on -debug "DB2.SQLB.sqlbPoolTblNewPool.entry" -suspend
Trace is turned on

$ db2 create tablespace ts1
<...hangs...>

SESSION 2
$ db2 connect to sample

$ db2 "list tablespaces"
<...hangs...>
```

db2pd –latches Example

```
$ db2pd -latches

Database Member 0 -- Active -- Up 0 days 00:22:36 -- Date 2018-10-23-18.15.09.261949

Latches:
Address            Holder Waiter Filename LOC  LatchType            HoldCount
0x0000000201FD0470 14     0      Unknown  1391 SQLO_LT_sqeWLDispatcher__m_tunerLatch 1
0x00007F890472B6F0 18     45     Unknown  2941 SQLO_LT_SQLB_PTBL__pool_table_latch 1
0x0000000202AA7C88 18     0      Unknown  526  SQLO_LT_preventSuspendIOLotch 1
```

- Look for lines with non-zero values in the “Waiter” column
- In this case, there is a contention on a “pool table latch”
  - Holder: EDU 18
  - Waiter: EDU 45

analyzestack Example

```
$ db2pd -stack all
Attempting to produce all stack traces for database partition.
See current DIAGPATH for stack trace file.

$ ~/sqllib/pd/analyzestack -i ~/sqllib/db2dump -l
**** 1 LATCHWAIT DETECTED ****
Please check the following files:

LatchAnalysis.out
******** LATCHWAIT DETECTED ( #1 )  ***********
<<<<   Holder Information (Address = 0x7f890472b6f0)   >>>>
   18 (/home/db2inst2/sqllib/db2dump/5274.18.000.stack.txt)
   Agent Type:  db2agent (SAMPLE)

<<<<   Waiter Information (Address = 0x7f890472b6f0)  >>>>
   TOTAL WAITERS >> 1
   45 (/home/db2inst2/sqllib/db2dump/5274.45.000.stack.txt)
   Agent Type:  db2agent (SAMPLE)
```

30

Look for lines containing “LATCHWAIT DETECTED”. In conjunction with the call stack files located in the diagnostic path, “LatchAnalysis.out” often contains enough information to determine the root cause:

- EDUs involved in the latch wait
- The call stacks of the respective EDUs (sequence of calls leading to the hang)
- Timestamps
- EDU types

db2fodc – First Occurrence Data Capture

- ***Situation:*** *You want to bring the data collection to the next level and collect the maximum amount of information for a subsequent problem determination*
- Solution:
  - `db2fodc -hang basic`
- Find `db2fodc -hang` too slow? No problem!
  1. Copy `~/sqllib/bin/db2cos_hang` to `~/sqllib/adm/db2cos_hang`
  2. In `~/sqllib/adm/db2cos_hang`, search for **`no_wait="OFF"`**
  3. Change to **`no_wait="ON"`**
  4. Execute `db2fodc` as usual

31

Under the hood, **db2fodc** executes a call-out script, in this case **db2cos_hang**. The script is located under "~/sqllib/bin", and cannot be modified by the instance owner. The solution is to copy the script to "~/sqllib/adm". The copy in "adm" takes precedence over the one in "bin". Once copied, the script in "adm" can be modified.

Most of the time consumed by **db2cos_hang** is spent by waiting in between data collection iterations. This is by design. Separating the data by a time offset gives an analyst a more accurate picture of the situation. However, when in a hurry, the wait times can be eliminated completely by changing the "no_wait" option of the script.

db2fodc –hang Example

```
$ db2fodc -hang basic
"db2fodc": List of active databases: "SAMPLE"

Starting data collection for hang problem determination...
Tue Oct 23 19:03:40 EDT 2018
...
Collecting OS Configuration info (started at 07:03:40 PM)
Should complete in less than one minute
Finished at 07:03:41 PM
...
Collecting DB2 CONFIG info (started at 07:04:16 PM)
Estimated time to completion is 5 minutes (Ctrl-C to interrupt)
Finished at 07:04:17 PM

Output directory is /home/db2inst2/sqllib/db2dump/FODC_Hang_2018-10-23-19.03.40.064993_0000
Open db2fodc_hang.log in that directory for details of collected data
```


- This run is done with `no_wait="ON"`
- The output data will be located in an FODC directory in the diagnostic path
- You can run the usual tools, such as **analyzestack**, on the output data in the FODC directory

What's in My Table Space?

- ***Situation:*** *You want to see how individual objects in your table space are laid out, and/or which object is holding the high water mark.*
- Solution:
  - `db2dart <dbname> /DHWM /TSI <tablespaceID>`
- Objects in a table space may be placed "all over" the table space
- There may be "holes", i.e. free space, anywhere in the table space
- The documentation claims that *"Practically speaking, it's virtually impossible to determine the high water mark yourself"*… we beg to differ! ☺

db2dart /DHWM Example

```
$ db2dart test /dhwm
High water mark:  538 pages, 269 extents (extents #0 - 268)

 [0000] 65534 0x0e   [0001] 65534 0x0e   [0002] 65535 0x00   [0003] == EMPTY ==
 [0004] == EMPTY ==  [0005] == EMPTY ==  [0006] == EMPTY ==  [0007] == EMPTY ==
 <...skipping...>
 [0132] == EMPTY ==  [0133] == EMPTY ==  [0134] == EMPTY ==  [0135] == EMPTY ==
 [0136]     5 0x40*  [0137]     5 0x00*  [0138]     5 0x43*  [0139]     5 0x03*
 [0140]     5 0x44*  [0141]     5 0x04*  [0142]     5 0x00   [0143]     5 0x00
 [0144]     5 0x00   [0145]     5 0x00   [0146]     5 0x00   [0147]     5 0x00
 <...skipping...>
 [0268]     5 0x00

Object holding high water mark:
 Object ID: 5
 Type:      Table Data Extent
```

Extent Number

Object ID

Object Type

Bed Time: DB2SLEEP

- ***Situation:*** *You are dealing with an outage (e.g. trap, data corruption, forced database shutdown), and you wish Db2 would freeze all processing instead of shutting down so you can still collect additional runtime information.*
- Solution:
  - `db2set DB2SLEEP=ON`
- Actions requiring Db2 engine processing (e.g. CONNECT, MON_GET*) will not be possible, but you will be able to use `db2pd, db2dart,…`
- To resume the shutdown, use `db2pcfg -wakeupinstance`

DB2SLEEP: Example

```
$ db2set DB2SLEEP=ON
$ db2stop;db2start
$ db2pd -edus
Database Member 0 -- Active -- Up 0 days 00:00:46 -- Date 2018-10-23-19.43.01.861562
List of all EDUs for database member 0
db2sysc PID: 17845

$ kill -SEGV 17845
$ kill -SEGV 17845

$ ls -d ~/sqllib/db2dump/FODC*
/home/db2inst2/sqllib/db2dump/FODC_Trap_2018-10-23-19.43.59.963061_0000

$ db2pd -edus
Database Member 0 -- Active -- Up 0 days 00:02:23 -- Date 2018-10-23-19.44.38.160836
List of all EDUs for database member 0
db2sysc PID: 17845
```

36

- In order to kill Db2, we are sending the SIGSEGV (Signal #11) to the db2sysc PID
- The signal needs to be sent twice because Db2 has its own signal handlers:
  - When Db2 receives a signal, Db2's own signal handlers first produce Db2 diagnostic data (e.g. FODC_Trap)
  - Then Db2 resets the signal handler to the OS default, and re-executes the same failing instruction, usually causing the process shutdown
- We can see that when DB2SLEEP is on, the db2sysc PID is still active
- At this point we can run additional non-engine (db2pd, db2dart, …) commands

BACKUP SLIDES

Common Db2 Component Prefixes

sql, squ	Backup and Restore
sqb	Buffer Pool Services: buffer pools, data storage management, table spaces, containers, I/O, prefetching, page cleaning
sqf	Configuration - database, database manager, configuration settings
sqd, sqdx, sqdl	Data Management Services: tables, records, long field and lob columns, REORG TABLE utility
sqp, sqdz	Data Protection Services: logging, crash recovery, rollforward
hdr	High Availability Disaster Recovery (HADR)
sqx	Index Manager
sqrl	Catalog Cache and Catalog Services
sqng	Code Generation (SQL Compiler)
squ, sqi, squs, sqs	Load, Sort, Import, Export
sqpl	Locking
sqno, sqnx, sqdes	Optimizer
sqo, sqz, oss	Operating System Services: AIX, Linux, Solaris, HP-UX platforms

Note the symbolic names use add an extra '1'. For example, sqlbAlterPoolAct from the previous example has the prefix of 'sqlb', which translates to component 'sqb' – buffer pool services.

Hangs: Important Routines

- The following routines serve as the first eyecatcher. An EDU executing these routines is always waiting for a latch, and this EDU should be closely examined:

  - `getConflictComplex`
  - `sqloltch`
  - `sqloltch_notrack`
  - `sqloSpinLockConflict`

Hangs: Less Important Routines

- The presence of the following routines usually (but not always ☺) indicates that the owning EDU is legitimately idle (e.g. sleeping, waiting for work), and the problem is elsewhere:
  - `msgrcv`
  - `ossSleep`
  - `semtimedop`
  - `sqleIntrptWait`
  - `sqloCSemP`
  - `sqloWaitEDUWaitPost`
  - `sqlorest`
  - `sqlorqueInternal`

- Also, if an application state is “UOW Waiting”, this application is NOT executing inside the Db2 kernel. Instead, the application is waiting for a remote request (usually outside of Db2) => not a Db2 issue.

Trap Signals/Exceptions

UNIX/Linux Signal ID	Description
SIGILL(4), SIGFPE(8), SIGTRAP(5), SIGBUS(10, Linux: 7), SIGSEGV(11), SIGKILL(9)	Instance trap. Bad programming, HW errors, invalid memory access, stack and heap collisions, problems with vendor libraries, OS problems. The instance shuts down.

Windows Exception	Description
ACCESS_VIOLATION (0xC0000005) ILLEGAL_INSTRUCTION (0xC000001D) INTEGER_DIVIDE_BY_ZERO (0xC0000094) PRIVILEGED_INSTRUCTION (0xC0000096) STACK_OVERFLOW (0xC00000FD)	Instance trap. Bad programming, HW errors, invalid memory access, stack overflows, problems with vendor libraries, OS problems. The instance shuts down.

•On UNIX, a signal can be sent to a Db2 process by issuing a "kill - <signal #>. Signals are defined in the "signals.h" header file.

•For example, on AIX 5.3, the signal.h header file is located in /usr/include.sys/signal.h

•An extract of the signal.h header file is as follows:

- #define SIGHUP 1 /* hangup, generated when terminal disconnects */
- #define SIGINT 2 /* interrupt, generated from terminal special char */
- #define SIGQUIT 3 /* (*) quit, generated from terminal special char */
- #define SIGILL 4 /* (*) illegal instruction (not reset when caught)*/
- #define SIGTRAP 5 /* (*) trace trap (not reset when caught) */
- #define SIGABRT 6 /* (*) abort process */
- #define SIGEMT 7 /* EMT intruction */
- #define SIGFPE 8 /* (*) floating point exception */
- #define SIGKILL 9 /* kill (cannot be caught or ignored) */
- #define SIGBUS 10 /* (*) bus error (specification exception) */
- ….
- ….

- To send an abort signal (SIGABRT) to a process, issue a "kill -6 <pid>".
- On Windows, use db2pd –stack to send "signals" to db2 processes/threads.

- **WARNING: DO NOT randomly issue signals to a Db2 process unless directed to by Db2 Service. Sending inappropriate signals can lead to database problems.**

Abort Signals/Exceptions

UNIX/Linux Signal IDs	Description
most UNIX's: SIGABRT(6) **HP-UX: SIGIOT(6)**	Instance panic. Self induced by Db2 due to unrecoverable problems. Typically associated with data (disk) corruption. The instance shuts down.

Windows Exception	Description
User Defined Exception (0xE0000002)	Diagnostic info signal. Dumps diagnostic info for the failing EDU. The instance shuts down during subsequent processing.

•On UNIX, a signal can be sent to a Db2 process by issuing a "kill - <signal #>. Signals are defined in the "signals.h" header file.

•For example, on AIX 5.3, the signal.h header file is located in /usr/include.sys/signal.h

•An extract of the signal.h header file is as follows:

- #define SIGHUP 1 /* hangup, generated when terminal disconnects */
- #define SIGINT 2 /* interrupt, generated from terminal special char */
- #define SIGQUIT 3 /* (*) quit, generated from terminal special char */
- #define SIGILL 4 /* (*) illegal instruction (not reset when caught)*/
- #define SIGTRAP 5 /* (*) trace trap (not reset when caught) */
- #define SIGABRT 6 /* (*) abort process */
- #define SIGEMT 7 /* EMT intruction */
- #define SIGFPE 8 /* (*) floating point exception */
- #define SIGKILL 9 /* kill (cannot be caught or ignored) */
- #define SIGBUS 10 /* (*) bus error (specification exception) */
- ….
- ….

- To send an abort signal (SIGABRT) to a process, issue a "kill -6 <pid>".
- On Windows, use db2pd –stack to send "signals" to db2 processes/threads.

- **WARNING: DO NOT randomly issue signals to a Db2 process unless directed to by Db2 Service. Sending inappropriate signals can lead to database problems.**

IDUG
Leading the DB2 User Community since 1988

IDUG EMEA Db2 Tech Conference
St. Julians, Malta | November 4 - 8, 2018

#IDUGDb2

Pavel Sustr
IBM Toronto Lab
psustr@ca.ibm.com
@pavel_sustr

Session code: C6

Please fill out your session evaluation before leaving!