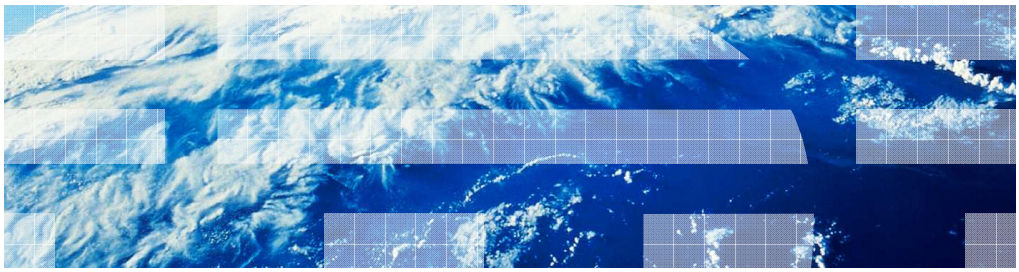
IBM InfoSphere DataStage

Operating system level tracing



Information Management

This presentation discusses the operating system utility tracing technique called “truss”.

Truss (1 of 6)

- Runs a command, then produces a trace of:
  - the system calls it performs
  - signals it receives
  - machine faults incurred
- Can also connect to a running process
- Can start and trace new processes such as the execution of “dsjob”

The truss utility executes the specified command and produces a trace of the system calls it performs, the signals it receives, and the machine faults incurred. Truss can also connect to a process that is already running. For example, you can truss the dsapi_slave process for an established client connection by connecting truss to the dsapi_slave process. You can also use truss to start and trace new processes such as the execution of “dsjob”.

Truss (2 of 6)

- Most often used arguments
  - **-p** <Process ID> - Connects truss to a process that is already running
  - **-o** <output file name> - Directs output to a specific file name
  - **-f** Follow all children processes created by fork() or vfork()

If you want to connect to an existing process, use the –p argument with the process ID of the process you want to trace. Use the –o argument to specify the name of the output file you want to write the trace information to. The –f argument is very important as it will continue to trace child processes created by fork or vfork.

Truss (3 of 6)

- Other useful arguments
  - **-a** Shows the argument strings that are passed in each exec() system call
  - **-e** Shows the environment strings that are passed in each exec() system call
- Example using truss on a dsapi_slave process with process ID 12345
  - truss –fae –p 12345 –o /tmp/truss.out
- Must use truss as root or as the owner of the process
  - Non-root users cannot truss another user's processes

Some other useful arguments are –a, which shows the argument strings that are passed in each exec() system call. The -e argument shows the environment strings that are passed in each exec() system call.

An example of using truss on a dsapi_slave process with a process ID of 12345 is:

truss –fae –p 12345 –o /tmp/truss.out.

You must use truss as root or as the owner of the process. Non-root users cannot truss another user's processes.

Truss (4 of 6)

- Where can you use truss?
  - Tracing odbc connection failures – drivers failing, not a database failure
  - Connect truss to dsapi_slave process to trace
    - Client disconnecting
    - Plugin not loading
    - Metadata import failing
  - Use truss for issues that can be reproduced easily

Some examples of places where truss is useful when supporting DataStage® include tracing odbc connection failures. It is also useful to connect truss to dsapi_slave process to trace things such as client disconnecting, plug-ins not loading and metadata import failing.

Truss outputs a large amount of data so you want to use it for easily reproducible issues and not for a problem where a job aborts after 50,000 rows are processed.

Truss (5 of 6)

- What will truss display?
  - Where truss is looking for libraries
  - What files are opened by the process
  - What other processes are executed
- What abilities does truss offer?
  - To see what is succeeding
  - To see what is failing
  - To see what files and libraries the process is using or searching for

Truss will show you things like where it is looking for libraries it needs to load, what files are opened by the process, what other processes are executed, and more. It allows you the ability to see what is succeeding and what is failing and what files and libraries the process is using or searching for.

Truss (6 of 6)

- Truss equivalents on other platforms
  - **strace** - Linux®
  - **trace** (32 bit) – HP-UX 10
    - This does not come with HPUX, however, you can get it on the Web
  - **tusc** (64 bit) – HP-UX 11
    - It is not installed by default. HP customers can request it for free from HP support
  - **truss -** AIX®
  - **strace** - NT
    - A “truss like” utility available on-line

This slide displays some examples of other “truss like” tracing tools on other platforms. You will want to look at the manual page to get the exact syntax for each.

Truss – example 1 (1 of 2)

- Cannot get odbc drivers to work
- Error data source cannot be found
- Added data source to the $DSHOME/.odbc.ini file and uvodbc.config
- Checked DSN; everything looks ok
- Source .dsenv file
- Run demoodbc and still receives error
- Use truss to see which .odbc.ini file it is looking at

This first example involves how you can use truss to resolve an issue where the odbc drivers are not connecting properly. In this example, you cannot connect to a database using odbc. You are getting an error message that the data source cannot be found. You have verified that the data source has been added to both the $DSHOME/.odbc.ini file and the uvodbc.config file in the project directory. You have checked the DSN in both files and everything looks correct. In order to trace properly, the first thing you want to do is source the dsenv file so that your environment is set up like the DataStage environment. Next, run the demoodbc program to see if that works or fails. In this example, you find that demoodbc gives the same error message as DataStage. Use truss to try to find the error. Since it is saying it can't find the data source and you know the data source is in the .odbc.ini file that was edited, use truss to see exactly which .odbc.ini file it is looking at.

Truss – example 1 (2 of 2)

- truss –fae –o /tmp/truss.out demoodbc MyOracle -UID scott -PWD tiger
- Search the truss.out file for “odbc.ini”

```
lwp_schedctl(SC_STATE|SC_BLOCK, -1, 0xEF383D1C) = 0
lwp_schedctl(SC_DOOR, 0, 0x00000000)            = 4
door_bind(4)                                    = 0
close(4)                                        = 0
close(3)                                        = 0
sigaction(SIGWAITING, 0xEF5E6258, 0x00000000)   = 0
lwp_self()                                      = 1
open("/export/home/Ascential/DataStage/branded_odbc/.odbc.ini", O_RDONLY)  = 3
```

- Not looking at the right .odbc.ini file
  - Check ODBCINI env variable in dsenv to point to correct file

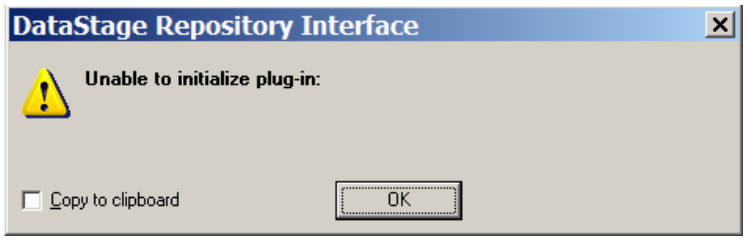
You are going to use truss to trace the demoodbc connection since this shows the same error and is a simpler process and will produce less output in your trace file. To do this type:

truss –fae –o /tmp/truss.out demoodbc MyOracle -UID scott -PWD tiger

Search the truss.out file for “odbc.ini”. This will show you the full path to the file that it opened. You can see it is not looking at the right .odbc.ini file. You need to check the ODBCINI environment variable in dsenv and make it point to the correct file. It should be looking at the .odbc.ini file located in the DSEngine directory and not the one in branded_odbc.

Truss – example 2 (1 of 7)

- Try to import Oracle plug-in metadata with ORAOCI9 plug-in
- Receive error:



In this second example, you receive an error when trying to import plug-in metadata with the Oracle plug-in. The error is “Unable to initialize plug-in”. Normally this error indicates an issue with loading a library that the plug-in requires but it isn’t possible to tell from this error message which library it is having problems with.

Truss – example 2 (2 of 7)

- How do you debug this?
  - Get the process ID of the dsapi_slave process

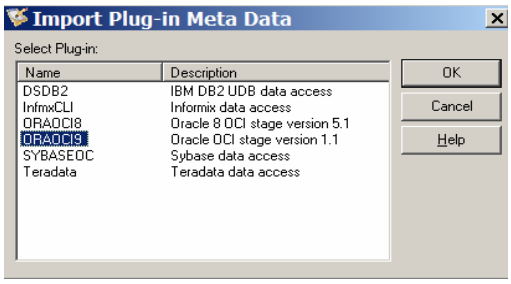
```
$ ps -ef|grep dscs
  dsadm  90028  79144   0 16:23:42      -  0:00 752dscs 4 7200 0
  dsadm  91150 103460   0 16:36:58 pts/13  0:00 grep dscs
  dsadm  96528  65032   0 11:21:15      -  0:00 234dscs 4 0 0
  dsadm 101208  79144   0 15:23:10      -  0:00 752dscs 4 7200 0
  dsadm 101974  79144   0 14:15:32      -  0:00 752dscs 4 7200 0
  dsadm 103024  79144   0 14:55:38      -  0:00 752dscs 4 7200 0
  dsadm 103698  79144   0 16:24:26      -  0:00 752dscs 4 7200 0
  dsadm 110888  79144   0 14:54:31      -  0:00 752dscs 4 7200 0
  dsadm 114518  82986   0   Oct 02      -  0:00 75adscs 4 0 0
  dsadm 116228  87772   0 16:33:40      -  0:00 dscs 4 28800 0
$ ps -ef|grep 116228
  dsadm  32596 116228   0 16:33:40      -  0:00 dsapi_slave 7 6 0
  dsadm  35078 103460   0 16:37:20 pts/13  0:00 grep 116228
  dsadm 116228  87772   0 16:33:40      -  0:00 dscs 4 28800 0
```


You need to attach truss to the running dsapi_slave process for the client that is receiving the error. In this example, your were using a DataStage instance that was running on the default port. Perform a ps –ef|grep dscs to see which dscs process is on the default port.

The process ID for dscs is 116228. Truss the child dsapi_slave process because dscs is the parent. Do another ps –ef|grep 116228 to find the child dsapi_slave process. In this case the process ID for the dsapi_slave process is 32596.

Truss – example 2 (3 of 7)

- Go to point right before problem



  - Attach truss to the running process
    - –$ truss –fae –o /tmp/truss.out -p 32596
    - –Continue until you see the error message

Because truss outputs a large volume of data, it is necessary to try to get as close to the error as possible before attaching truss to the process. This will help minimize the amount of output. Since the error occurs when you are importing metadata, bring up that screen first and then attach the truss. The command to use is:

truss –fae –o /tmp/truss.out -p 32596

When you enter this command, you will not get a prompt back, that's ok, let it sit. Next, continue with your import until you see the error message.

Truss – example 2 (4 of 7)

- Error received
  - Type ^c to exit the truss
- Check log file
  - 75% of the log file is unreadable
  - Look for readable content
    - What files are being opened
    - Where do open files fail
    - Where do open files succeed
    - What error messages produced

Once you have received the error from the client, go back to the window where the truss is running and type “control C”. This will exit the truss process. Once it is complete, look at the log file.

Most of the truss file can be information that is not readable by the average user. Look for things that can easily be read, such as what files are being opened, where do the open files fail, where do they succeed and what error messages are produced.

Truss – example 2 (5 of 7)

- Successful opening of library libc.so.1

```
1347:   stat("/ext2/ds751A/Ascential/DataStage/DSEngine/java/jre/lib/sparc/clien
1347:   stat("/ext2/ds751A/Ascential/DataStage/DSEngine/java/jre/lib/sparc/libc.
so.1", 0xFFBFEE8C) Err#2 ENOENT
1347:   stat("/ext2/ds751A/Ascential/DataStage/branded_odbc/lib/libc.so.1", 0xFF
1347:   stat("/ext2/ds751A/Ascential/DataStage/DSEngine/lib/libc.so.1", 0xFFBFEE
1347:   stat("/ext2/ds751A/Ascential/DataStage/DSEngine/uvdlls/libc.so.1", 0xFFB
1347:   stat("/opt/oracle/ora92/lib42/libc.so.1", 0xFFBFEE8C) Err#2 ENOENT
1347:   stat("/ext2/SYBASE/OCS-12_5/lib/libc.so.1", 0xFFBFEE8C) Err#2 ENOENT
1347:   stat("/usr/lib/libc.so.1", 0xFFBFEE8C)          = 0
```


The output displayed on this slide is what you see when you are able to successfully open the library libc.so.1. What you see in the trace is the process going through each directory in the library path until either the file is found or there are no more directories in the library path and the search fails. In this case, the library is found in /usr/lib.

Truss – example 2 (6 of 7)

- Problem found

```
1338:   stat("/ext2/ds751A/Ascential/DataStage/DSEngine/java/jre/lib/sparc/clien
t/libclntsh.so", 0xFFBFD50C) Err#2 ENOENT
1338:   stat("/ext2/ds751A/Ascential/DataStage/DSEngine/java/jre/lib/sparc/libcl
ntsh.so", 0xFFBFD50C) Err#2 ENOENT
1338:   stat("/ext2/ds751A/Ascential/DataStage/branded_odbc/lib/libclntsh.so", 0
xFFBFD50C) Err#2 ENOENT
1338:   stat("/ext2/ds751A/Ascential/DataStage/DSEngine/lib/libclntsh.so", 0xFFB
FD50C) Err#2 ENOENT
1338:   stat("/ext2/ds751A/Ascential/DataStage/DSEngine/uvdlls/libclntsh.so", 0x
FFBFD50C) Err#2 ENOENT
1338:   stat("/opt/oracle/ora92/lib42/libclntsh.so", 0xFFBFD50C) Err#2 ENOENT
1338:   stat("/ext2/SYBASE/OCS-12_5/lib/libclntsh.so", 0xFFBFD50C) Err#2 ENOENT
1338:   stat("/usr/lib/libclntsh.so", 0xFFBFD50C)       Err#2 ENOENT
1338:   munmap(0xFE34A000, 4504)                       = 0
1338:   munmap(0xFE300000, 239904)                     = 0
```


As you continue to look through the trace file, you can see that the process is looking for the libclntsh.so library. You can see truss going through all of the directories in your library path but it never finds the libclntsh.so library. You know that libclntsh.so is an Oracle client library. If it is a library that you don't know, do a "find" on your server to see where the library is located.

Truss – example 2 (7 of 7)

- Typing error made in the library path is found
  - stat("/opt/oracle/ora92/lib42/libclntsh.so", 0xFFBFD50C) Err#2 ENOENT
- The user needs to fix the typing error in dsenv and stop and restart the engine

For this example, if you look closely at the trace file, you can see that it was looking for the library in "/opt/oracle/ora92/**lib42**". You can see that there is a typing error in the library path, it should be /opt/oracle/ora92/lib**32 not 42**. In this case, it is necessary to update the dsenv file, fix the typing error in the library path, and then stop and restart the DataStage engine.

Feedback

Your feedback is valuable

You can help improve the quality of IBM Education Assistant content to better meet your needs by providing feedback.

- Did you find this module useful?
- Did it help you solve a problem or answer a question?
- Do you have suggestions for improvements?

Click to send e-mail feedback:

mailto:iea@us.ibm.com?subject=Feedback_about_DS_operating_sys.ppt

This module is also available in PDF format at: ../DS_operating_sys.pdf

You can help improve the quality of IBM Education Assistant content by providing feedback.

Trademarks, disclaimer, and copyright information