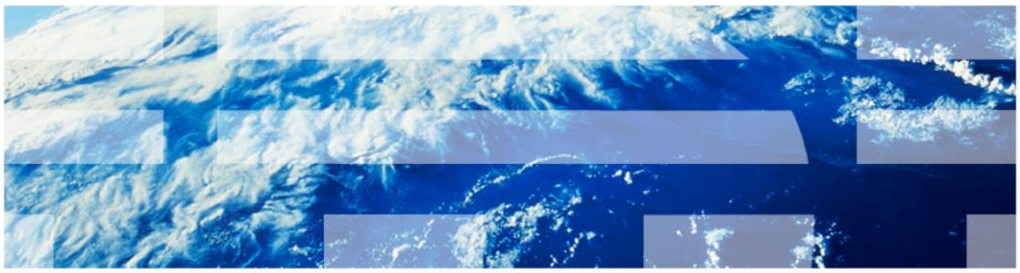
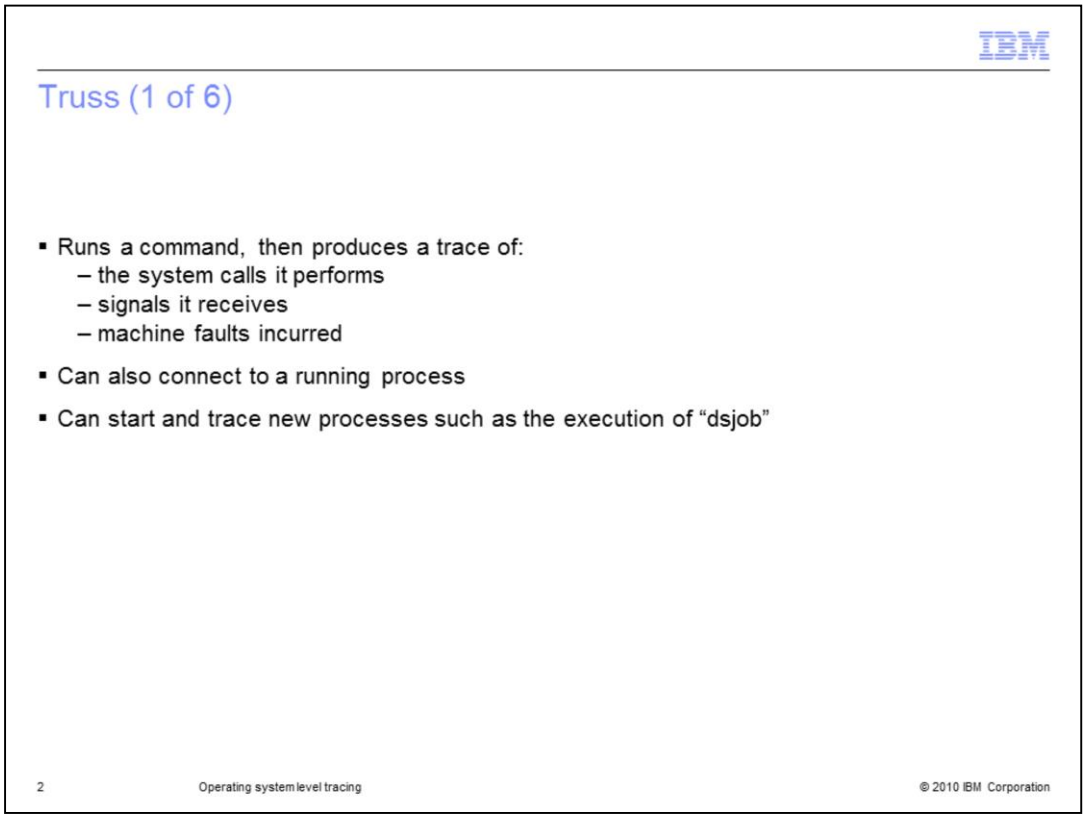
IBM

IBM InfoSphere DataStage

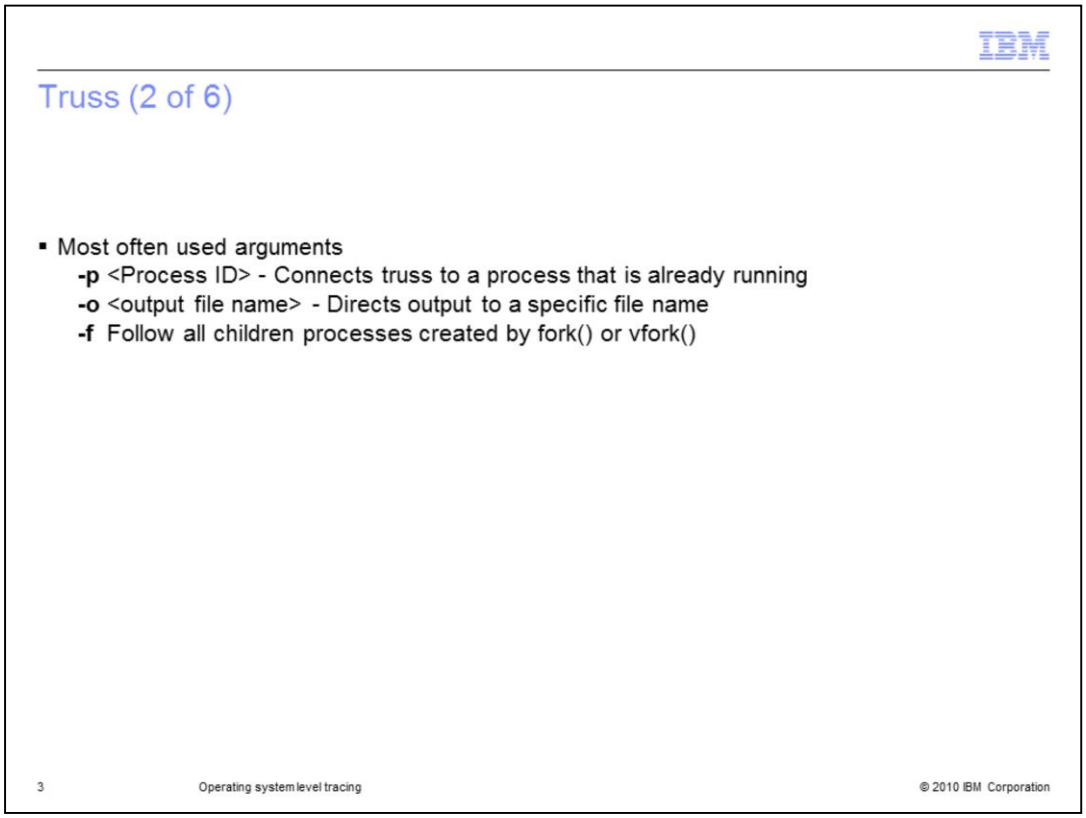
Operating system level tracing



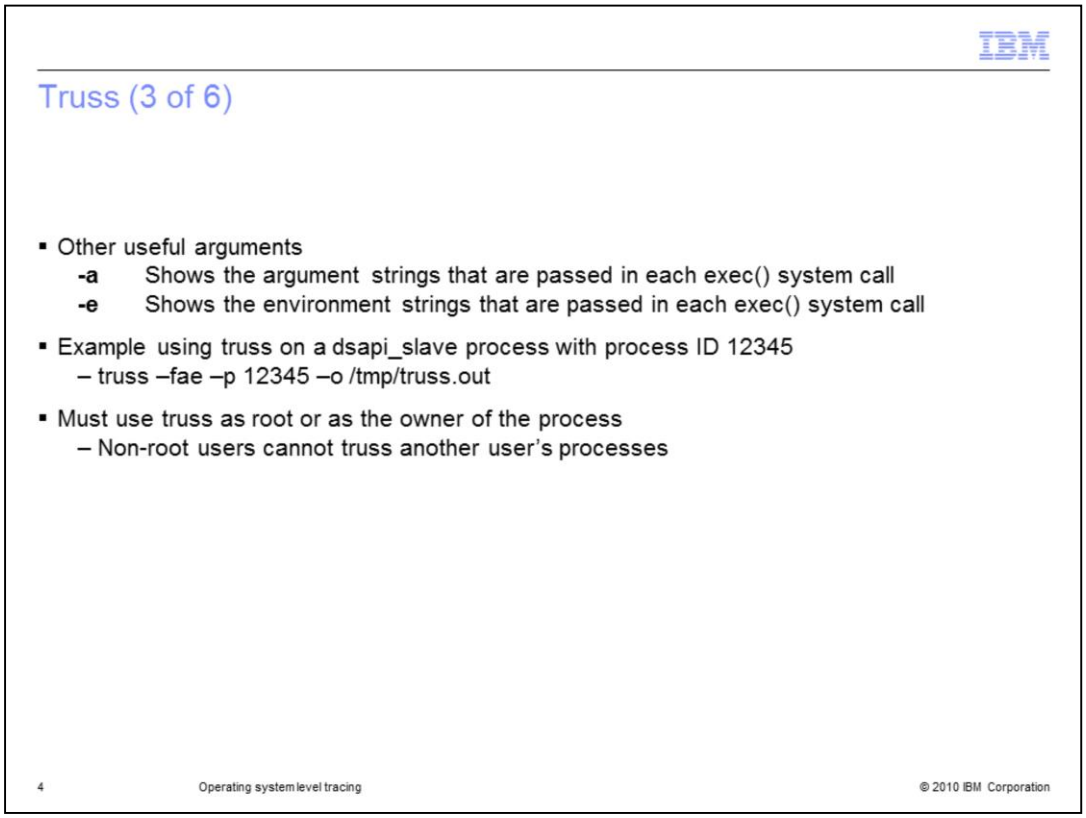
This presentation discusses the operating system utility tracing technique called “truss”.



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”.



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.

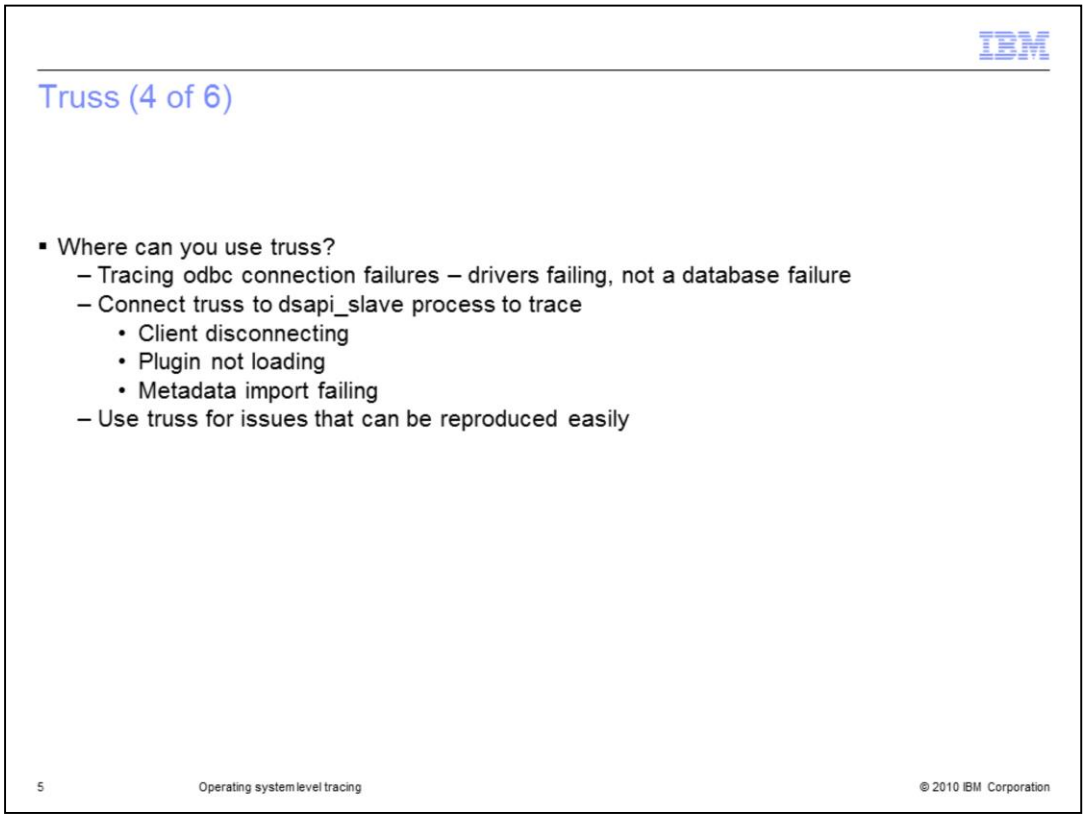


Some other useful arguments are –a, which shows the argument strings that are passed in each exec() system call. The -e argument shows the environment strings that are passed in each exec() system call.

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

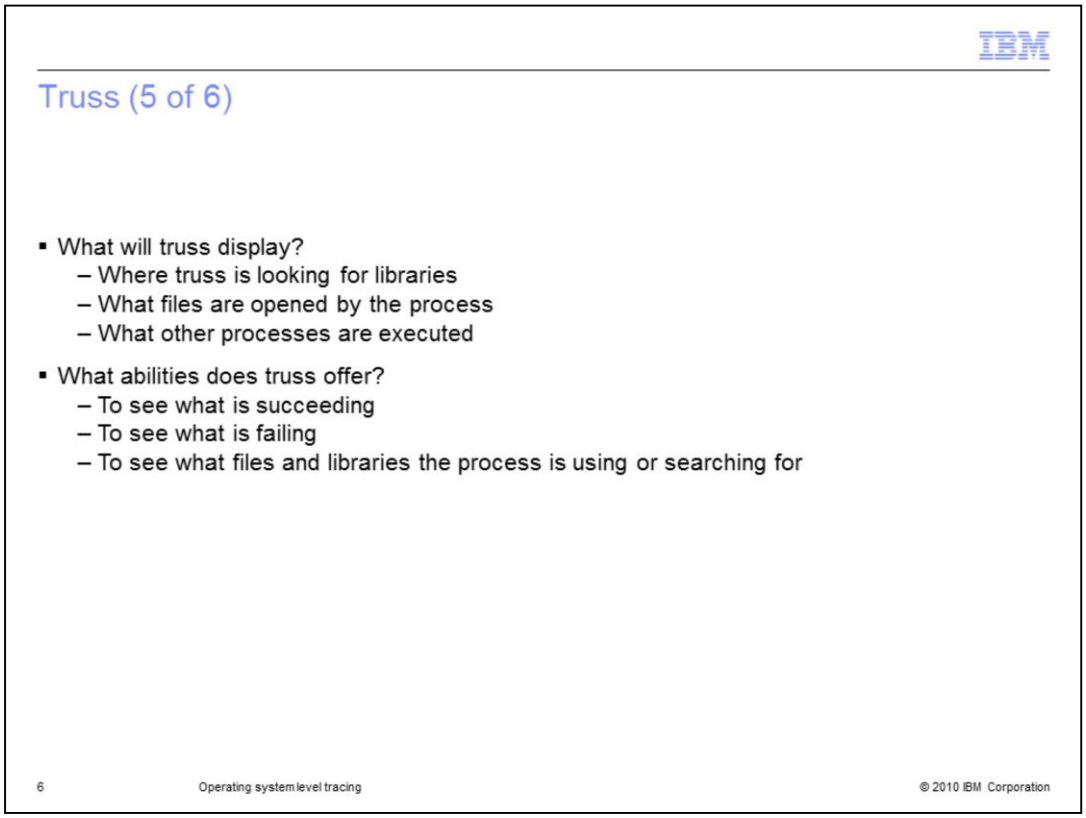
truss –fae –p 12345 –o /tmp/truss.out.

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

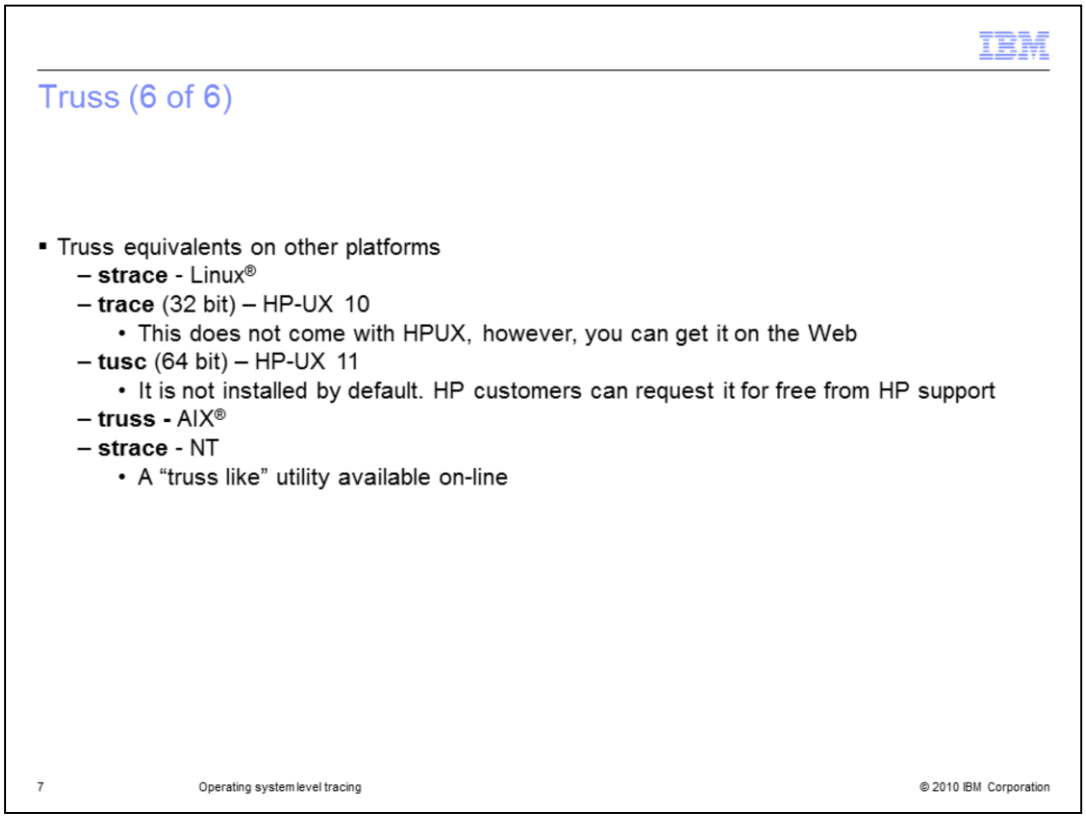


Some examples of places where truss is useful when supporting DataStage® include tracing odbc connection failures. It is also useful to connect truss to dsapi_slave process to trace things such as client disconnecting, plug-ins not loading and metadata import failing.

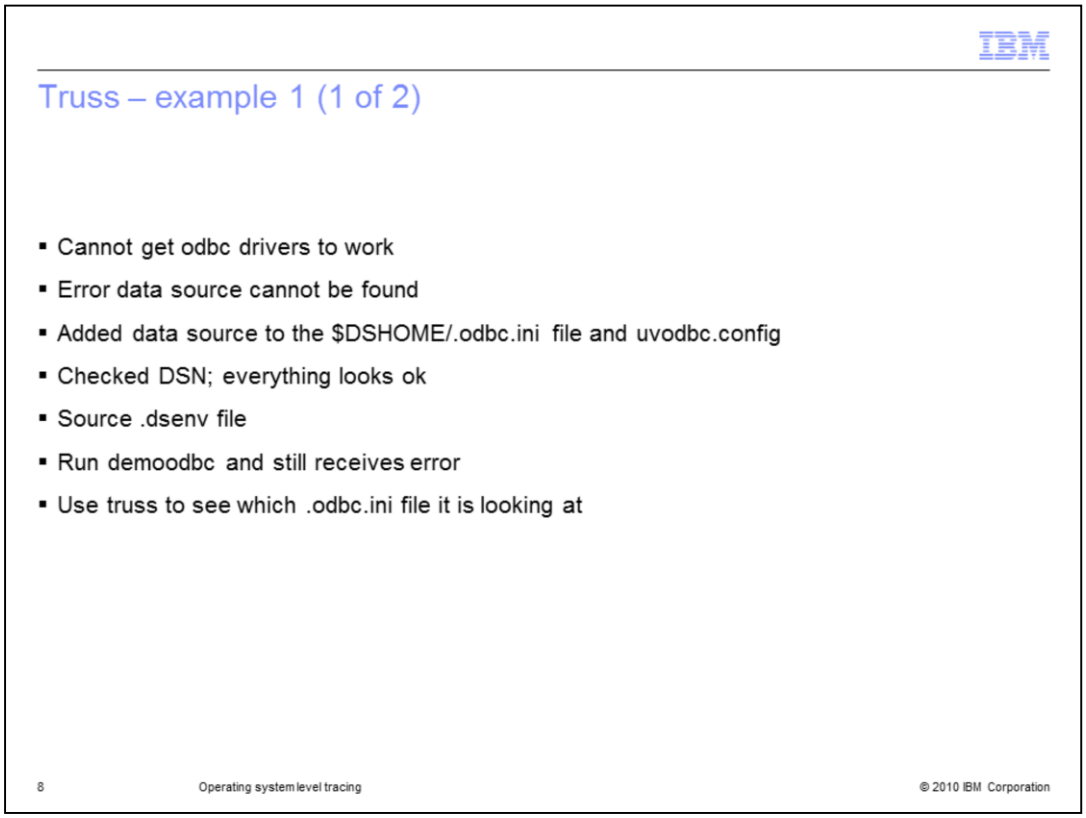
Truss outputs a large amount of data so you want to use it for easily reproducible issues and not for a problem where a job aborts after 50,000 rows are processed.



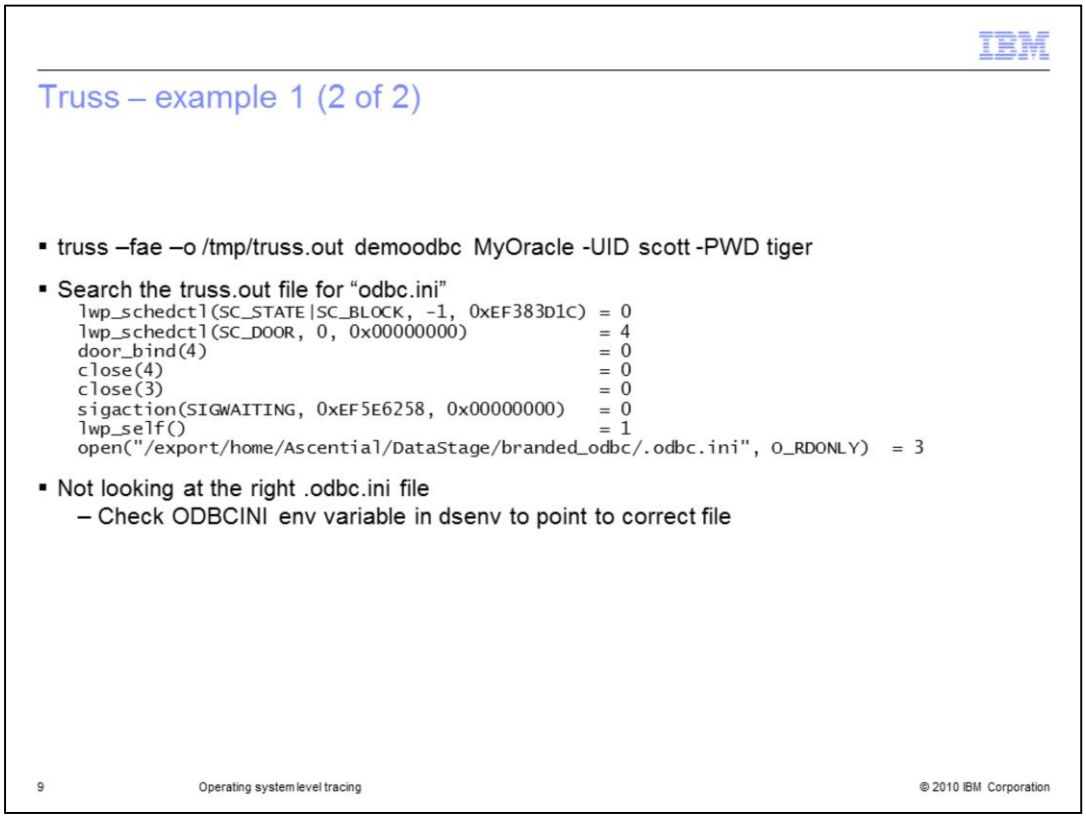
Truss will show you things like where it is looking for libraries it needs to load, what files are opened by the process, what other processes are executed, and more. It allows you the ability to see what is succeeding and what is failing and what files and libraries the process is using or searching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



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.



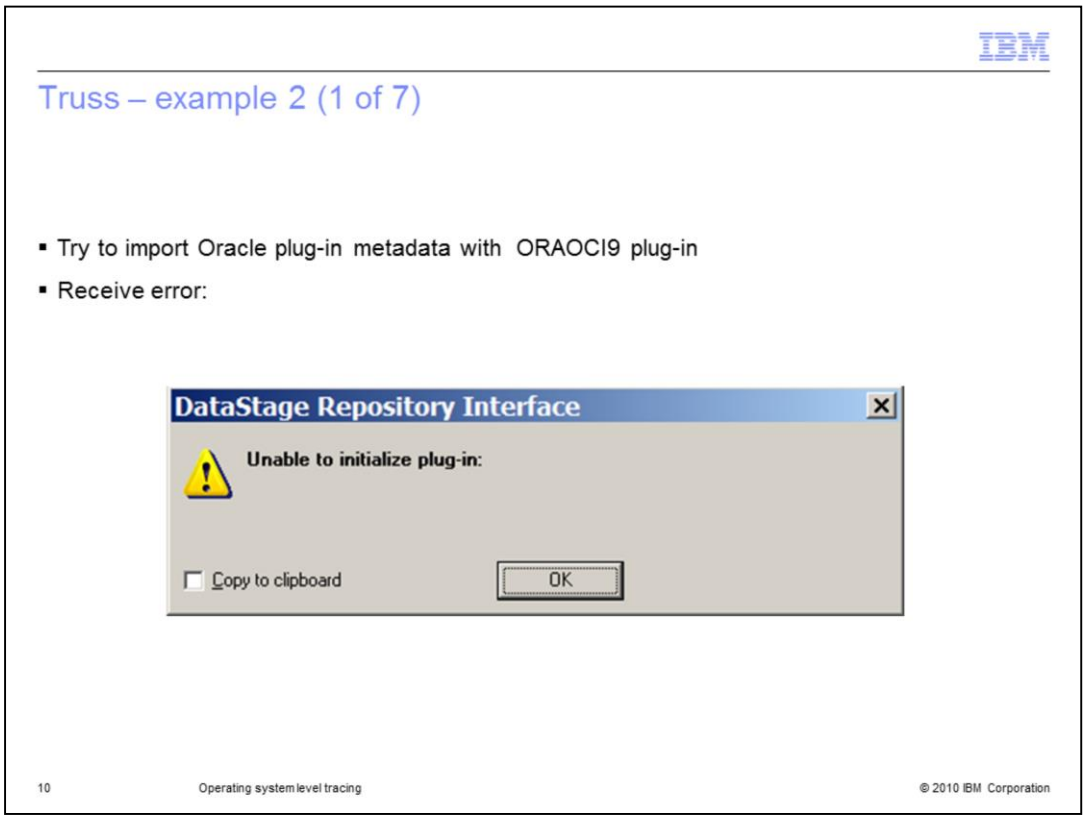
This first example involves how you can use truss to resolve an issue where the odbc drivers are not connecting properly. In this example, you cannot connect to a database using odbc. You are getting an error message that the data source cannot be found. You have verified that the data source has been 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.



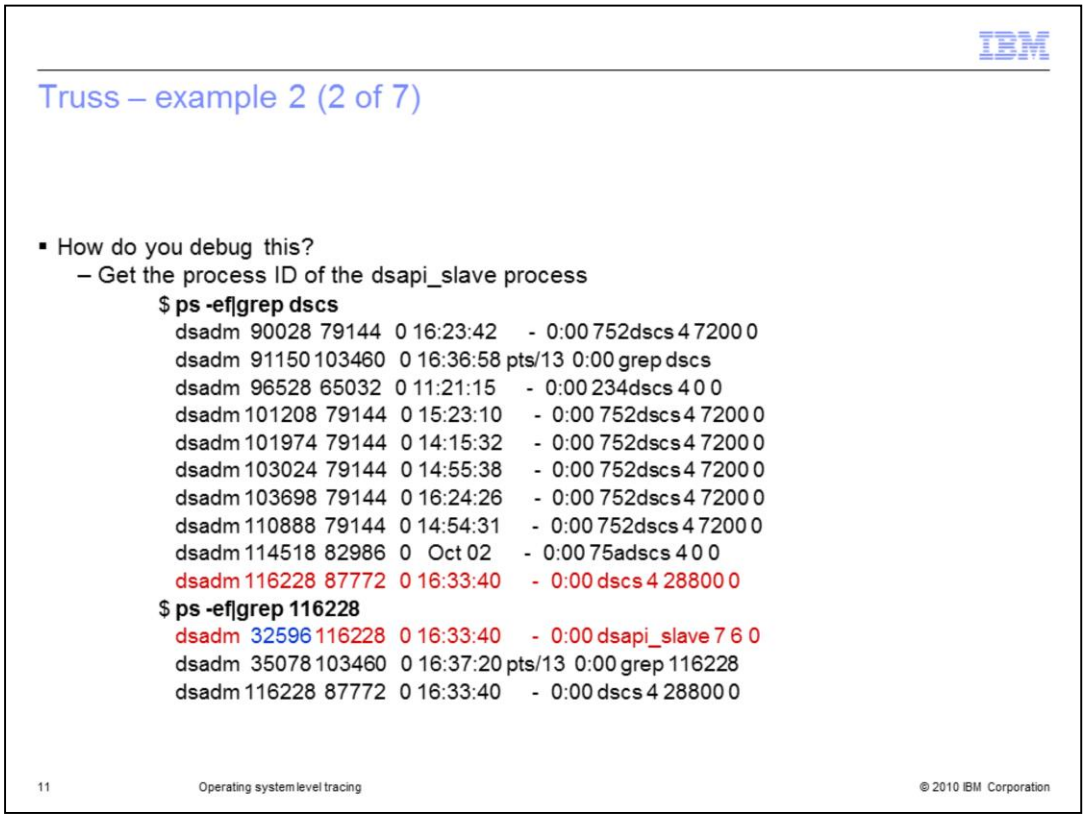
You are going to use truss to trace the demoodbc connection since this shows the same error and is a simpler process and will produce less output in your trace file. To do this type:

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.



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.

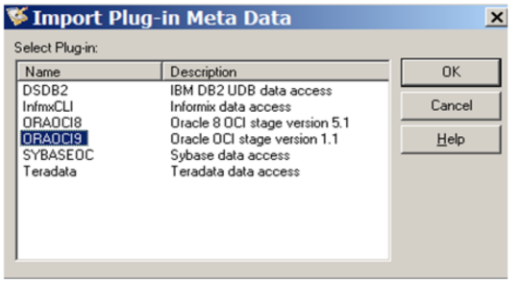


You need to attach truss to the running dsapi_slave process for the client that is receiving the error. In this example, your were using a DataStage instance that was running on the default port. Perform a ps –ef|grep dscs to see which dscs process is on the default port.

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



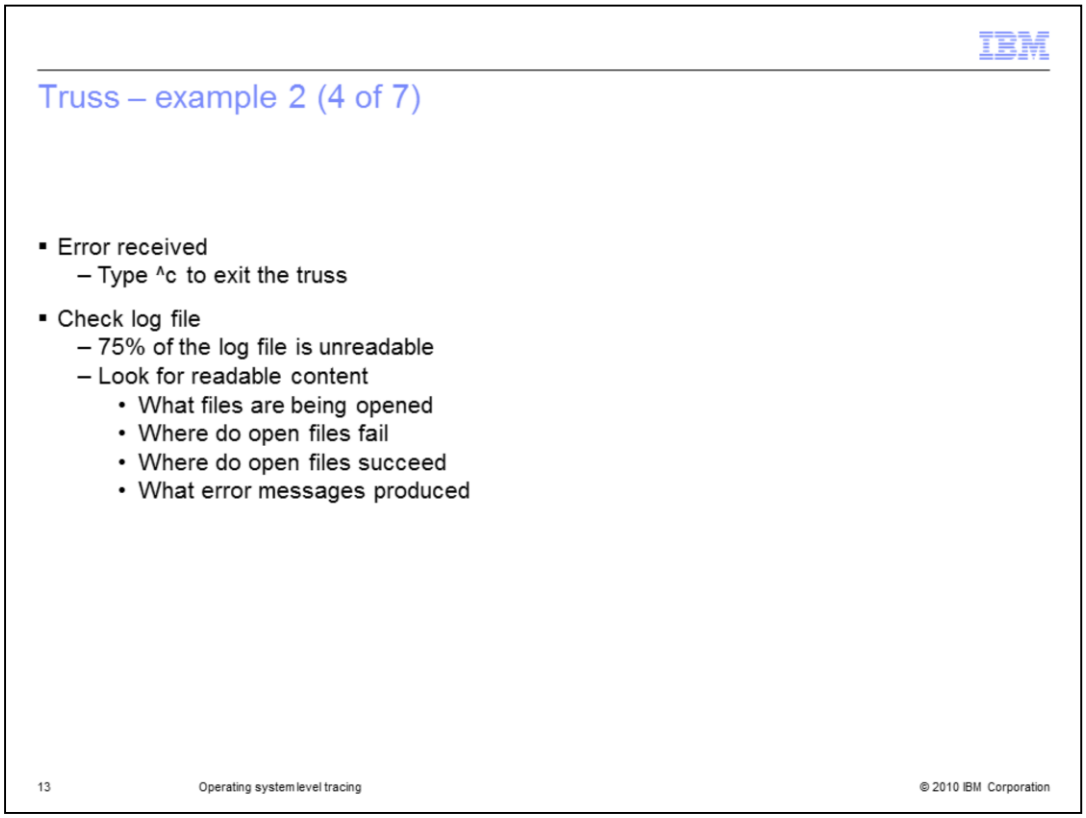
- Attach truss to the running process
  - –$ truss –fae –o /tmp/truss.out -p 32596
  - –Continue until you see the error message

12 Operating system level tracing

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:

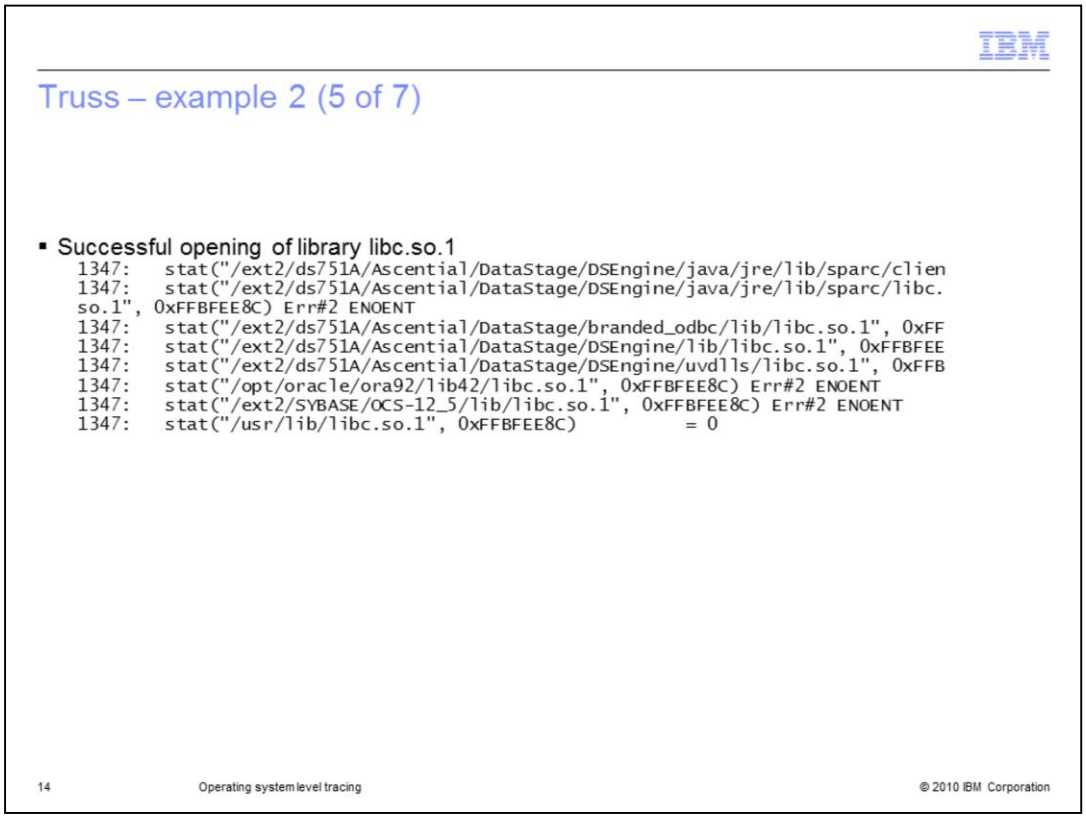
truss –fae –o /tmp/truss.out -p 32596

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.

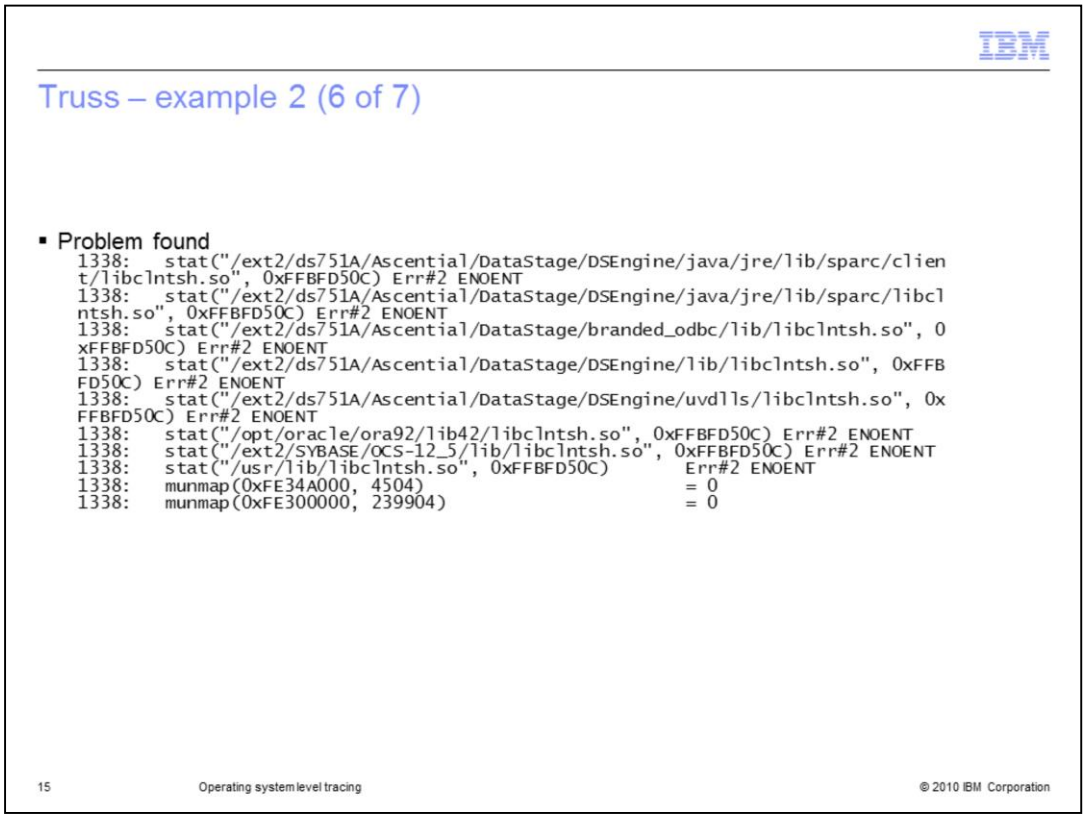


Once you have received the error from the client, go back to the window where the truss is running and type “control C”. This will exit the truss process. Once it is complete, look at the log file.

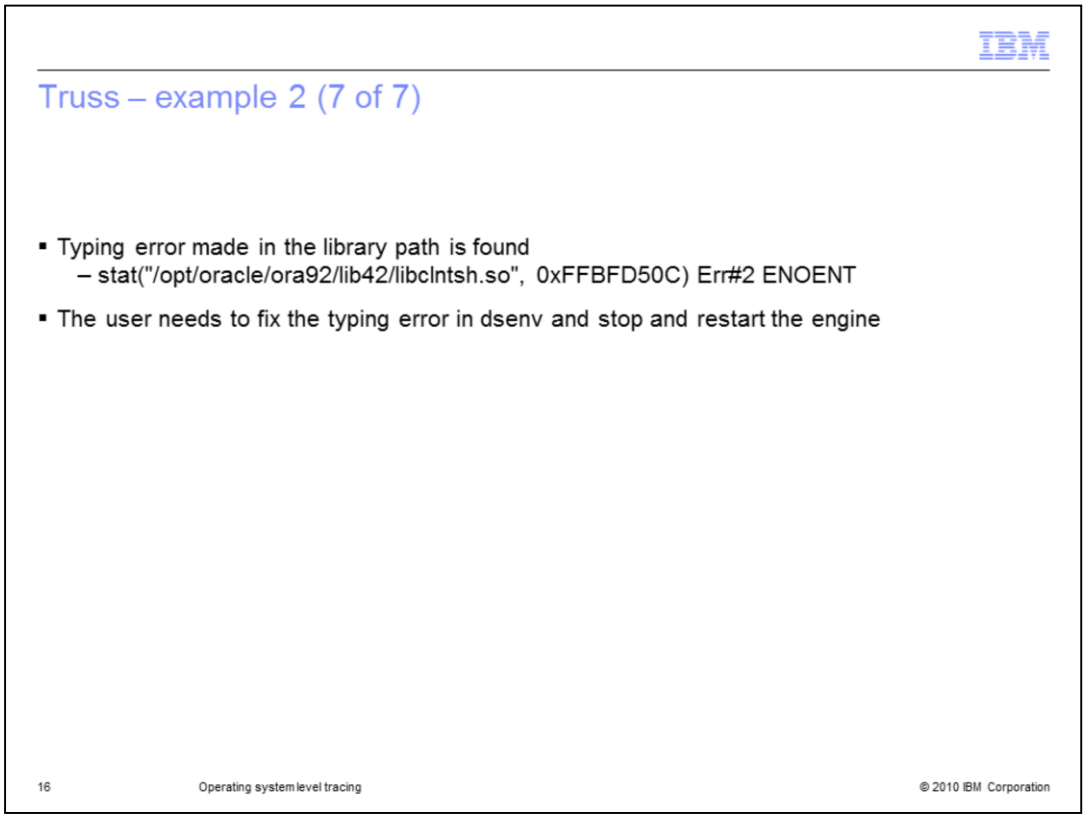
Most of the truss file can be information that is not readable by the average user. Look for things that can easily be read, such as what files are being opened, where do the open files fail, where do they succeed and what error messages are produced.



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.



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.



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.

Trademarks, disclaimer, and copyright information