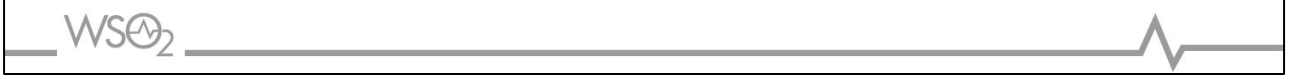

WSO2 Product Administration
WSO2 Training
Module 08 - Troubleshooting
CC BY 4.0

Troubleshooting

- Understanding Log Files
- Remote Debugging
- Analyzing stack trace
- Capturing the state of the system
- Viewing process threads in solaris
- Network packet capture
- Thread usage



Running Deployments may encounter various issues.

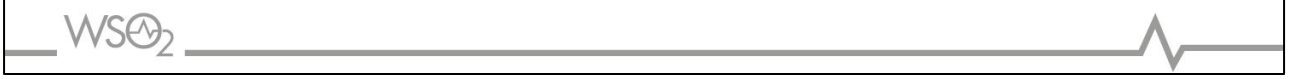
Evidence, causes and clues for these issues can be identified via logs, debugging and with the use of various tools.

It is important to have an idea about the level of information exposed by various means of troubleshooting and how to troubleshoot.

https://wso2.com/library/webinars/2018/04/troubleshooting-and-best-practices-with-wso2-enterprise-integrator/

Understanding Log Files

- Log files expose server activity and errors
- Used for troubleshooting
- Knowledge on the level at which information related to the investigated problem is captured
    - Trace logs
    - Wire logs
    - Debug logs
- To identify incorrect/missing information in messages, messages traversing through incorrect flows, incorrect handling of messages etc.



Exploring and investigating log files is the entry point to troubleshooting issues.
To investigate an issue using log files, it is also important to have an understanding on the type of log file that exposes the relevant logs.

For example, investigating debug logs is one of the basic steps to retrieve further information on an issue.
You have to enable debug logs for different packages depending on the requirement.
Carbon_HOME/repository/conf/log4j.properties file governs logging on the server as explained previously, and it can be used to enable/disable various loggers based on the information that need to be captured in carbon logs.

E.g., To debug an issue in a mediation flow :

Enable debug logs for the entry 'log4j.category.org.apache.synapse'
i.e. log4j.category.org.apache.synapse=DEBUG

If synapse transport level debug logs need to be enabled specifically, enable debug logs for the entry 'log4j.category.org.apache.synapse.transport'
Apart from these there are trace logs that trace the mediation path of a

message, wire logs that capture the messages sent over http/https transports etc.

By looking at these logs it should be possible to identify the root causes, such as missing information, incorrect message flows and incorrect handling of messages

Remote Debugging

Start servers in Debug mode

```
sh wso2server.sh -debug <port>
```

WSO2 products start up supports the start-up option '--debug', which starts up the server instance in the debug mode.

This enables remote connections to the server via an exposed port.

To start the server in the debug mode you can execute the command 'sh wso2server.sh -debug <port>' in the <Product_home>/bin/ directory.

You can connect to the server using the debug port via an IDE.

To investigate the code execution flows and identify the erroneous scenarios, you can either repeat the scenarios on the UI, or simulate similar conditions using the IDE itself.

Analyzing Stack Trace

1. jstack <pid> > thread-dump.txt
2. ps -C java -L -o pcpu,cpu,nice,state,cputime,pid,tid > thread-usage.txt

thread_usage.txt

TID with highest %CPU → 1604

```
%CPU CPU  NI S     TIME   PID   TID
.........
 0.0   -   0 S 00:00:00  1519  1602
 0.0   -   0 S 00:00:00  1519  1603
24.8   -   0 R 00:06:19  1519  1604
 2.4   -   0 S 00:00:37  1519  1605
 0.0   -   0 S 00:00:00  1519  1606
.........
```

thread_dump.txt

1604 in hexadecimal → 644

```
"HTTPS-Sender I/O dispatcher-1" prio=10 tid=0x00007fb54c010000 nid=0x644 runnable [0x00007fb534e20000]
   java.lang.Thread.State: RUNNABLE
        at org.apache.http.impl.nio.reactor.IOSessionImpl.getEventMask(IOSessionImpl.java:139)
        - locked <0x00000006cd91fef8> (a org.apache.http.impl.nio.reactor.IOSessionImpl)
        at org.apache.http.nio.reactor.ssl.SSLIOSession.updateEventMask(SSLIOSession.java:300)
        at org.apache.http.nio.reactor.ssl.SSLIOSession.inboundTransport(SSLIOSession.java:462)
        - locked <0x00000006cd471df8> (a org.apache.http.nio.reactor.ssl.SSLIOSession)
        at org.apache.http.impl.nio.reactor.AbstractIODispatch.inputReady(AbstractIODispatch.java:121)
        at org.apache.http.impl.nio.reactor.BaseIOReactor.readable(BaseIOReactor.java:160)
        at org.apache.http.impl.nio.reactor.AbstractIOReactor.processEvent(AbstractIOReactor.java:342)
        at org.apache.http.impl.nio.reactor.AbstractIOReactor.processEvents(AbstractIOReactor.java:320)
        at org.apache.http.impl.nio.reactor.AbstractIOReactor.execute(AbstractIOReactor.java:280)
        at org.apache.http.impl.nio.reactor.BaseIOReactor.execute(BaseIOReactor.java:106)
        at org.apache.http.impl.nio.reactor.AbstractMultiworkerIOReactor$Worker.run(AbstractMultiworkerIOReactor.java:604)
        at java.lang.Thread.run(Thread.java:722)
```

Analysing thread dumps is important when it comes to troubleshooting issues related to high memory and CPU usage.
The following commands can be used to obtain a thread dump and a thread usage information to identify threads with high CPU usage.

To get a thread dump :
jstack <pid> thread_dump.txt

To get thread usage information:
ps -C java -L -o pcpu,cpu,nice,state,cputime,pid,tid > thread-usage.txt

thread-usage.txt obtained using the above commands will contain CPU usage % against the Thread ID.
Thread ID is represented in hexadecimal in the thread_dump.txt.
Convert the thread ID to hexadecimal and search for it in the thread_dump.txt.
thread_dump.txt will contain the stack trace of against the searched thread ID.

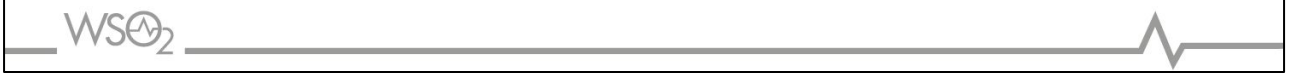
Capturing the State of the System

Carbondump tool to analyze the system

sh carbondump.sh [-carbonHome path] [-pid of the carbon instance]

You can create a **heap dump** and **thread dump** using the **CarbonDump** tool.

These will also provide information about the **product version** and **patch inconsistencies**.



Carbondump tool can be used to collect data from a running WSO2 product instance when an error occurs.

This generates a ZIP archive containing the necessary data that can be used to analyse the system and identify the issue causes.

This can be run using the following command against the carbon home.path and the carbon instance process ID.

sh carbondump.sh [-carbonHome path] [-pid of the carbon instance]

View Process Threads in Solaris

To verify whether process is

- Parallelized
- Taking advantage of threading capabilities of CPU

Run command

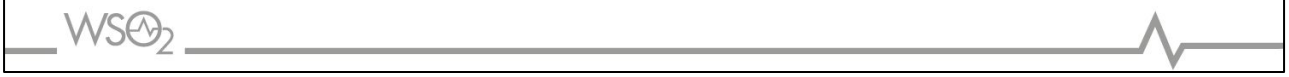
prstat

OR

prstat -L -p <pid>

To view individual thread activity

```
   PID USERNAME  SIZE   RSS STATE  PRI NICE      TIME  CPU PROCESS/NLWP
...
 12905 root     4472K 3640K cpu0    59    0   0:00:01 0.4% prstat/1
 18403 monitor   474M  245M run     59   17   1:01:28 9.1% java/103
  4102 oracle     12G   12G run     59    0   0:00:12 4.5% oracle/1
```



When database processes do not utilize the CPU's threading capabilities at its optimum level in Solaris, to figure out whether a Solaris process is parallelized and is taking advantage of the threading capabilities of the CPU.

For this,Run 'prstat' command in the command prompt of Solaris.
The NLWP value in the last column indicates the lightweight processes/number of threads the process is currently using with Solaris.
There is a one to one mapping between lightweight process and user threads.

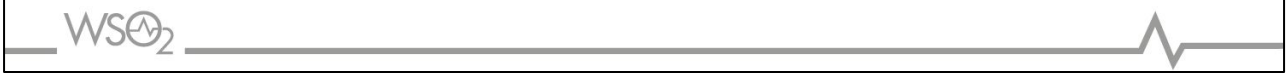
You can also view the individual thread activity of a multi threaded process using the options '-L' and '-p'.
eg : prstat -L -p <pid>
This will output the threads sorted by the CPU activity.
In this output the last column (PROCESs/LWPID) will indicate the LWPID which is the thread ID.
Here, if more than one thread shows significant activity, you can conclude that the process is actively taking advantage of multi-threading.

Network Packet Capture

tcpdump can be used to capture or filter TCP/IP packets that received or transferred over a network on a specific interface.

To capture and save the file in a .pcap format, execute command with -w option.

```
tcpdump -w 0001.pcap -i eth0
```



tcpdump is a most powerful and widely used command-line packets sniffer or package analyzer tool which is used to capture or filter TCP/IP packets that received or transferred over a network on a specific interface.

http://soatutorials.blogspot.com/2014/12/12-useful-tcpdump-commands-you-can-use.html

Thread Usage

Number of threads initiated and how much of the CPU is consumed by each thread

```
/bin/ps -C java -L -o
pcpu,cpu,nice,state,cputime,pid,tid
>thread_usage.txt
```

%CPU	CPU	NI	S	TIME	PID	TID
0.0	-	0	S	00:00:00	27668	27668
0.0	-	0	S	00:00:02	27668	27669
0.3	-	0	S	00:05:01	27668	27670
0.3	-	0	S	00:05:01	27668	27671
2.6	-	0	S	00:39:15	27668	27672
2.4	-	0	S	00:36:17	27668	27673
0.0	-	0	S	00:00:03	27668	27674
0.0	-	0	S	00:00:06	27668	27675

THANK YOU

wso2.com