Runtime Infrastructure - Support #3162

JVM crash in native code called from FileStream

08/05/2016 08:18 AM - Greg Shah

Status:	New	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:		**% Done:**	0%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**case_num:**	
vendor_id:	GCD	**version:**	

Description

History

#1 - 08/05/2016 11:54 AM - Greg Shah

- File hs_err_pid7691.log added

In June 2015, a client's automated test suite was encountering appserver agent JVM crashes in native code when called from the P2J FileStream constructor.

Eric dug into this and found that the server.log had these:

[06/12/2015 10:35:40 XXX] (com.goldencode.p2j.util.Agent:WARNING) Agent encountered an error while executing a command for appserver app_server
com.goldencode.p2j.net.SilentUnwindException: Connection ended abnormally
at com.goldencode.p2j.util.TransactionManager.honorStopCondition(TransactionManager.java:3728)
at com.goldencode.p2j.util.TransactionManager.popScope(TransactionManager.java:2371)
at com.goldencode.p2j.util.Agent.invokeScoped(Agent.java:199)
at com.goldencode.p2j.util.Agent.access$600(Agent.java:54)
at com.goldencode.p2j.util.Agent$4.execute(Agent.java:751)
at com.goldencode.p2j.util.Agent.listen(Agent.java:353)
at com.goldencode.p2j.util.AgentPool.start(AgentPool.java:431)
at com.goldencode.p2j.util.AppServerManager.startAppServer(AppServerManager.java:828)
at com.goldencode.p2j.main.StandardServer.standardEntry(StandardServer.java:256)
at sun.reflect.GeneratedMethodAccessor379.invoke(Unknown Source)
at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
at java.lang.reflect.Method.invoke(Method.java:606)
at com.goldencode.p2j.util.MethodInvoker.invoke(MethodInvoker.java:76)
at com.goldencode.p2j.net.Dispatcher.processInbound(Dispatcher.java:693)
at com.goldencode.p2j.net.Conversation.block(Conversation.java:319)
at com.goldencode.p2j.net.Conversation.run(Conversation.java:163)
at java.lang.Thread.run(Thread.java:745)

For each of the server log entries, Eric found a matching JVM crash logs (hs_err_pid*.log). Each one showed the same basic stack trace:

Java frames: (J=compiled Java code, j=interpreted, Vv=VM code)
j java.lang.ClassLoader$NativeLibrary.load(Ljava/lang/String;)V+0
j java.lang.ClassLoader.loadLibrary1(Ljava/lang/Class;Ljava/io/File;)Z+302
j java.lang.ClassLoader.loadLibrary0(Ljava/lang/Class;Ljava/io/File;)Z+2
j java.lang.ClassLoader.loadLibrary(Ljava/lang/Class;Ljava/lang/String;Z)V+217
j java.lang.Runtime.loadLibrary0(Ljava/lang/Class;Ljava/lang/String;)V+54
j java.lang.System.loadLibrary(Ljava/lang/String;)V+7
j sun.security.action.LoadLibraryAction.run()Ljava/lang/Void;+4
j sun.security.action.LoadLibraryAction.run()Ljava/lang/Object;+1
v ~StubRoutines::call_stub
j java.security.AccessController.doPrivileged(Ljava/security/PrivilegedAction;)Ljava/lang/Object;+0
j sun.nio.ch.Util.load()V+41
j sun.nio.ch.IOUtil.<clinit>()V+17
v ~StubRoutines::call_stub
j sun.nio.ch.Util.<clinit>()V+17
v ~StubRoutines::call_stub
j sun.nio.ch.FileChannelImpl.<clinit>()V+29
v ~StubRoutines::call_stub

```
j  java.io.RandomAccessFile.getChannel()Ljava/nio/channels/FileChannel;+26
j  com.goldencode.p2j.util.FileStream.<init>(Ljava/lang/String;ZZ)V+94
j  com.goldencode.p2j.util.StreamDaemon.openFileStream(Ljava/lang/String;ZZ)I+33
v  ~StubRoutines::call_stub
j  sun.reflect.NativeMethodAccessorImpl.invoke0(Ljava/lang/reflect/Method;Ljava/lang/Object;[Ljava/lang/Object;)Ljava/lang/Object;+0
j  sun.reflect.NativeMethodAccessorImpl.invoke(Ljava/lang/Object;[Ljava/lang/Object;)Ljava/lang/Object;+87
j  sun.reflect.DelegatingMethodAccessorImpl.invoke(Ljava/lang/Object;[Ljava/lang/Object;)Ljava/lang/Object;+6
j  java.lang.reflect.Method.invoke(Ljava/lang/Object;[Ljava/lang/Object;)Ljava/lang/Object;+57
j  com.goldencode.p2j.util.MethodInvoker.invoke([Ljava/lang/Object;)Ljava/lang/Object;+9
j  com.goldencode.p2j.net.Dispatcher.processInbound(Lcom/goldencode/p2j/net/InboundRequest;ZLcom/goldencode/p2j/net/NetResource;)V+480
j  com.goldencode.p2j.net.Conversation.block()Lcom/goldencode/p2j/net/Message;+110
j  com.goldencode.p2j.net.Conversation.waitMessage(I)Lcom/goldencode/p2j/net/Message;+1
j  com.goldencode.p2j.net.Queue.transactImpl(Lcom/goldencode/p2j/net/Message;I)Lcom/goldencode/p2j/net/Message;+214
j  com.goldencode.p2j.net.Queue.transact(Lcom/goldencode/p2j/net/Message;I)Ljava/lang/Object;+31
j  com.goldencode.p2j.net.BaseSession.transact(Lcom/goldencode/p2j/net/Message;I)Ljava/lang/Object;+11
j  com.goldencode.p2j.net.HighLevelObject.transact(Lcom/goldencode/p2j/net/RoutingKey;[Ljava/lang/Object;)Ljava/lang/Object;+17
j
com.goldencode.p2j.net.RemoteObject$RemoteAccess.invokeCore(Ljava/lang/Object;Ljava/lang/reflect/Method;[Ljava/lang/Object;)Ljava/lang/Object
;+47
j  com.goldencode.p2j.net.InvocationStub.invoke(Ljava/lang/Object;Ljava/lang/reflect/Method;[Ljava/lang/Object;)Ljava/lang/Object;+139
j  com.sun.proxy.$Proxy5.standardEntry(Lcom/goldencode/p2j/main/ClientParameters;)Z+16
j  com.goldencode.p2j.main.ClientCore.start(Lcom/goldencode/p2j/cfg/BootstrapConfig;ZZ)V+317
j  com.goldencode.p2j.main.ClientCore.start(Lcom/goldencode/p2j/cfg/BootstrapConfig;Z)V+3
j  com.goldencode.p2j.main.ClientDriver.start(Lcom/goldencode/p2j/cfg/BootstrapConfig;)V+28
j  com.goldencode.p2j.main.CommonDriver.process([Ljava/lang/String;)V+170
j  com.goldencode.p2j.main.ClientDriver.process([Ljava/lang/String;)V+2
j  com.goldencode.p2j.main.ClientDriver.main([Ljava/lang/String;)V+14
v  ~StubRoutines::call_stub
```

One of the hotspot crash log files which Eric found is attached for reference. They are all basically the same. There was one such crash log for each disconnect/abend in the server.log and the timestamps in the log matched the modification times of those files.

At the time, this problem could only be recreated on the customer's system AND it was intermittent. But strangely enough, when it did occur, it was always from the same FileStream constructor.

The native stack trace showed that the failure occurred when trying to load a native library required by sun.nio.ch.IOUtil, which is a pretty fundamental J2SE support class.

My assessment at the time:

The failure has nothing to do with our native code. It really doesn't have much to do with our Java code either. Our last code that is involved is the FileStream constructor. It fails on the call to getChannel().

```
   public FileStream(String filename, boolean write, boolean append)
   throws ErrorConditionException
   {
      String errmsg = null;
      int    errnum = 0;

      try
      {
         File   target = new File(filename);
         String mode   = (write ? "rw" : "r");

         // create a traditional file system resource with the proper mode
         RandomAccessFile raf = new RandomAccessFile(target, mode);

         // obtain the backing channel (we no longer need access to the raf)
         file = raf.getChannel();
```

This is not something that should be catastrophic for the JVM. The failure itself is in the Linux loader (ld-linux-x86-64.so.2):

```
Stack: [0x00007f85db3c1000,0x00007f85db4c2000],  sp=0x00007f85db4bc4b8,  free space=1005k
Native frames: (J=compiled Java code, j=interpreted, Vv=VM code, C=native code)
C  [ld-linux-x86-64.so.2+0x19c72]
```

The failure is a "trap e" in Intel x86 terms (memory violation), in Linux it is reported as SIGSEGV (segment violation). It means that the Linux loader code (or something downstream from it) accessed memory at an virtual address that was not mapped into the current process. In other words, it tried to read or write outside of its address space.

Of course, it could be as simple as being passed an invalid pointer (e.g. 0x0) by the caller, though I would have expected the Linux loader code to be safe from null pointer references. Most likely it is something much more tricky in that code.

Regardless, it is either something in the environment/config/setup of the installed Linux system OR it is buggy loader code in the installed Linux system or it is buggy JVM code that calls the loader. I am quite confident that it is not in P2J.

Some details about the customer environment:

- Ubuntu 14.04 running in a VM on VMWare ESX host.
- Java version "1.7.0_79"
- OpenJDK Runtime Environment (IcedTea 2.5.5) (7u79-2.5.5-0ubuntu0.14.04.2)
- OpenJDK 64-Bit Server VM (build 24.79-b02, mixed mode)

The same JDK version on Ubuntu 14.04 on Golden Code systems did not duplicate the issue, ever (as far as we know).

Absent a bug in the JVM implementation, it shouldn't be possible for pure Java code to cause this sort of an error. But clearly something P2J was doing in that environment exposed such a defect.

It was also noted that once this happened in one agent JVM, the other agents also seemed more likely to abend at the same place. So:

1. Something about the state kept in the P2J server (and presumably shared with follow-on clients) could have been exacerbating the problem, such that it occured more frequently after the initial failure. OR

2. Some state in the shared library (linux loader in this case) was corrupted and the appserver agent JVMs were sensitive to this bad state.

To get a better idea of what the converted 4GL code was doing at the time of the crash, Eric instrumented the FileStream constructor. The files being read were small files that contained delimited data like this:

```
CBX|false|true|false|false|false|
    CBT|false|true|false|false|false|
    CBS|false|true|false|false|false|
    CBD|false|true|false|false|false|
    CBM|false|true|false|false|false|
    CBV|false|true|false|false|false|
```

The corresponding logging from the FileStream constructor showed this:

```
FileStream c'tor:  filename = &lt;a_local_file_whose_contents_are_shown_above&gt;; write = false; append = fal
se
```

The file was being used for read purposes only. There were no security/permission problems with the file.

The customer then tried some things and reported:

> JVM bug reports and forum messages for problems similar to ours imply that setting LD_BIND_NOW=1 would workaround a fault in glibc which occurs in AVX support in various processors (including the Xeon processors used in our VM hosts). I've tried this and still get the crashes.

> I tried disabling compressed oops, which hasn't helped. Since trying that, I've found the following:

http://mail-archives.apache.org/mod_mbox/cassandra-user/201109.mbox/%3CCAAnh3__fKo6GgnL2F0KMx5Xy18rbDpLxGgJEpWcTH+qm_r-+aA@mail.gmail.com%3E
... which is very interesting as it involves use of memory mapped files, a lot like in the p2j.util.FileStream class, although the stacktrace is quite different to ours. Unfortunately I can't find the oracle or openjdk bug reports that they refer to, so I don't know whether the problems have been resolved.

> I'm seeing the same problems using the oracle jdk as I was using openjdk (perhaps not surprising since I think there is a lot of shared code between these projects).

> I've asked ICT about hardware diagnostics and had this response:

>> "We don't run memtest on our hosts, however vSphere displays realtime hardware status and alerts for all components in each blade/server. We also have realtime health monitoring at the physical level provided via the UCS interface. We haven't seen issues with any other systems so I think the hardware is ok."

On September 2, 2015 the customer reported that they had not seen the problem in a number of runs that suggested something was changed. They had made some changes but did not know which may have made the difference:

They moved to P2J rev 10930 which substantially reduced memory usage.

They tried some shared memory settings for the kernel. The report:

> The shared memory settings seems the most likely candidate for the fix, so I've investigated first by setting a kernel.shmmax to approx 4mb. The P2J server process starts, P2J client JVMs crash on starting. Next I set kernel.shmmax to 25mb and ran the full search tests. I expected to see client JVM crashes on reading from .d files and did not. I have now reduced shmmax to approx 15mb and I'm running the tests again - I'm hoping to see hotspot errors!

I doubt this was the actual fix.

I have no further record of discussions on this issue. As far as I know, the problem was not seen again AND we really don't know the reason. The changes in P2J may have modified the runtime environment enough that it bypassed the issue.

As noted above, the problem was also seen on Oracle's JDK so it is not a simple swap of JDK to resolve the problem.

#2 - 08/05/2016 11:58 AM - Greg Shah

- File hs_err_pid3022.log added

Today Constantin reported this:

-------- Forwarded Message --------
Subject: Re: Test questions
Date: Fri, 5 Aug 2016 14:00:15 +0300
From: Constantin Asofiei
To: Eric Faulhaber
CC: Igor Skornyakov , Ovidiu Maxiniuc , Greg Shah

Guys,

With batchProgramTest/3109c, all clients abend at some point with something like this (same stacktrace was with trunk/11076 and openjdk, too). Have you seen this before? There are no errors in client/server P2J logs.

Thanks,
Constantin

```
j  sun.nio.ch.NativeThread.current()J+0
j  sun.nio.ch.NativeThreadSet.add()I+0
J 4811 C1 sun.nio.ch.FileChannelImpl.write([Ljava/nio/ByteBuffer;II)J
(285 bytes) @ 0x00007f9ce939fc4c [0x00007f9ce939f820+0x42c]
J 4810 C1 com.goldencode.p2j.util.FileStream.writeFully([Ljava/nio/ByteBuffer;[II)V
(130 bytes) @ 0x00007f9ce946892c [0x00007f9ce94687a0+0x18c]
J 3974 C2 com.goldencode.p2j.util.FileStream.writeWorker([BII)V (162
```

```
bytes) @ 0x00007f9ce9ebe9d0 [0x00007f9ce9ebe4a0+0x530]
J 4770 C2 com.goldencode.p2j.util.Stream.flushControlled(Z)I (296 bytes)
@ 0x00007f9ce96022e8 [0x00007f9ce9602100+0x1e8]
J 4809 C2 com.goldencode.p2j.ui.chui.ThinClient.downWorker(IIZZZ)Z (254
bytes) @ 0x00007f9ce9929050 [0x00007f9ce9928b80+0x4d0]
J 4821 C2 sun.reflect.GeneratedMethodAccessor15.invoke(Ljava/lang/Object;[Ljava/lang/Object;)Ljava/lang/Object
;
(367 bytes) @ 0x00007f9ce9e0bedc [0x00007f9ce9e0bc40+0x29c]
J 3659 C2 com.goldencode.p2j.net.Dispatcher.processInbound(Lcom/goldencode/p2j/net/InboundRequest;ZLcom/golden
code/p2j/net/NetResource;)V
(789 bytes) @ 0x00007f9ce9d06be0 [0x00007f9ce9d065a0+0x640]
j com.goldencode.p2j.net.Conversation.block()Lcom/goldencode/p2j/net/Message;+123
j com.goldencode.p2j.net.Conversation.waitMessage(I)Lcom/goldencode/p2j/net/Message;+1
j com.goldencode.p2j.net.Queue.transactImpl(Lcom/goldencode/p2j/net/Message;I)Lcom/goldencode/p2j/net/Message;
+214
j com.goldencode.p2j.net.Queue.transact(Lcom/goldencode/p2j/net/Message;I)Ljava/lang/Object;+31
j com.goldencode.p2j.net.BaseSession.transact(Lcom/goldencode/p2j/net/Message;I)Ljava/lang/Object;+11
j com.goldencode.p2j.net.HighLevelObject.transact(Lcom/goldencode/p2j/net/RoutingKey;[Ljava/lang/Object;)Ljava
/lang/Object;+17
j com.goldencode.p2j.net.RemoteObject$RemoteAccess.invokeCore(Ljava/lang/Object;Ljava/lang/reflect/Method;[Lja
va/lang/Object;)Ljava/lang/Object;+47
j com.goldencode.p2j.net.InvocationStub.invoke(Ljava/lang/Object;Ljava/lang/reflect/Method;[Ljava/lang/Object;
)Ljava/lang/Object;+138
j com.sun.proxy.$Proxy5.standardEntry()Z+9
j com.goldencode.p2j.main.ClientCore.start(Lcom/goldencode/p2j/cfg/BootstrapConfig;ZZ)V+345
j com.goldencode.p2j.main.ClientCore.start(Lcom/goldencode/p2j/cfg/BootstrapConfig;Z)V+3
j com.goldencode.p2j.main.ClientDriver.start(Lcom/goldencode/p2j/cfg/BootstrapConfig;)V+28
j com.goldencode.p2j.main.CommonDriver.process([Ljava/lang/String;)V+170
j com.goldencode.p2j.main.ClientDriver.process([Ljava/lang/String;)V+2
j com.goldencode.p2j.main.ClientDriver.main([Ljava/lang/String;)V+14
```

#3 - 08/05/2016 12:05 PM - Greg Shah

As can be seen from the hotspot log, the failure is again coming from FileStream usage of J2SE features and the actual native code that fails is in ld-linux-x86-64.so.2:

```
Stack: [0x00007f9cffb29000,0x00007f9cffc2a000],  sp=0x00007f9cffc23e58,  free space=1003k
Native frames: (J=compiled Java code, j=interpreted, Vv=VM code, C=native code)
C  [ld-linux-x86-64.so.2+0x1ad12]
```

It seems highly likely this is the same issue. We know this is a JVM issue or possibly a Linux loader bug that is triggered by the native code in the

JVM.

Files

hs_err_pid7691.log	70 KB	08/05/2016	Greg Shah
hs_err_pid3022.log	65.5 KB	08/05/2016	Greg Shah