Using NSD – A Practical Guide

Overview

Introduction

This lab is intended to provide IBM Lotus Domino® Administrators an idea of the data that can be used in examining server crash and performance issues. Needless to say, a complete analysis of NSD is not possible within the allotted time. This lab focuses on using Call Stack data and Memcheck data perform a preliminary analysis and get you started in the right direction.

Contents

This lab contains the following topics:

<table>
<thead>
<tr>
<th>Topic</th>
<th>See Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Section I: NSD Basics</td>
<td>2-6</td>
</tr>
<tr>
<td>Section II: Call Stacks</td>
<td>7-13</td>
</tr>
<tr>
<td>Section III: Memcheck – Shared Memory</td>
<td>14-26</td>
</tr>
<tr>
<td>Section IV: Memcheck – Process Memory</td>
<td>27-33</td>
</tr>
<tr>
<td>Section V: Memcheck – Resource Usage Summary</td>
<td>34-36</td>
</tr>
<tr>
<td>Section VI: Correlating NSD Output</td>
<td>36-40</td>
</tr>
<tr>
<td>Section VII: NSD Checklist</td>
<td>41</td>
</tr>
<tr>
<td>Section VIII: Case Studies</td>
<td>42-47</td>
</tr>
<tr>
<td>Appendix: Student Answer Key</td>
<td>48-68</td>
</tr>
</tbody>
</table>

Original vs. Newer Format

Within the ND7 timeframe, NSD has undergone many format and behavioral changes. Many of these changes have been back ported to ND6.x (as of ND6.5.5). Within this lab, we refer to both formats, where the original format in ND6 is applies to versions 6.5.4 and earlier, and the newer format applies to versions 6.5.5 and later, including ND7.x. Where appropriate, we indicate the different KEYWORDs that can be used for searching within an NSD in the Newer vs. Original Format.

Please refer to page 3 for more discussion on the NSD back port for existing versions of ND6.x and ND7.x, referred to as the NSD Update Strategy.

Focus on Domino Server

Due to time constraints, this lab is focused on troubleshooting Domino Server crash and hang/performance issues. However, many of these techniques can also be applied to IBM Lotus Notes® Client crashes and hangs. Please feel free to discuss these additional troubleshooting techniques with the lab instructors.
Section I – NSD Basics

What is NSD? NSD (Notes System Diagnostic) is one of the primary diagnostics used for the Lotus Domino Product Suite. It is used to troubleshoot crashes, hangs, and severe performance problems for such products as:

- Domino Server & Notes Client
- Quickplace, DomDoc, Domino Workflow
- Sametime

In ND6 and ND7, NSD is used on all Notes Client and Domino Server Platforms except for the Apple Macintosh® platform.

- On UNIX – NSD is a shell script (nsd.sh), with Memcheck compiled as a separate binary
- On W32 – NSD is a compiled binary (nsd.exe), with Memcheck built into nsd.exe
- On iSeries – ND6 is a compiled binary. The source for NSD differs both in nature and in output from the other platforms. However, in ND7, NSD on iSeries has been modified to more closely match the output of NSD on the other platforms.

NSD Basics NSD can be run under two contexts:

1). Manually - run from a command prompt to troubleshoot hangs and performance problems

2). Automatically as part of Fault Recovery to troubleshoot crashes – this is enabled in server document (enabled by default)

The NSD.EXE/NSD.SH file is located in the program directory. When running NSD manually, in order for NSD to run properly, you must run NSD from the desired data directory. With no switches, NSD will collect full set of data, creating a log file in the IBM_TECHNICAL_SUPPORT directory with the following filename format:

- ND6 - nds_all_<Platform>_<Host>_MM_DD@HH_MM.log
- ND7 - nds_<Platform>_<ServerName>_YYYY_MM_DD@HH_MM_SS.log

Continued on next page
Section I – NSD Basics, Continued

Between ND6 and ND7, NSD underwent numerous important changes, including format changes to better represent the data, as well as behavioral changes to resolve previous issues and improve NSD’s reliability as a diagnostic tool.

In order to allow existing customers to take advantage of improvements to NSD in their current environments, IBM has made available an updated version of NSD for existing versions of Domino on Windows® 32-bit, IBM AIX®, SUN Solaris®, and Linux®. This updated version includes most of the features available in ND7.x.

The NSD Update Strategy means that IBM will periodically provide an NSD Update with the latest set of NSD fixes/enhancements for existing versions of Domino. This is done so that IBM Support and customers can leverage the latest fixes and enhancements for First Failure Data Collection. This will be done for existing versions of ND6.x as well as ND7.x. For more information regarding the NSD Update Strategy, including a list of fixes to NSD, refer to the following URL:


You may also refer to a recently published Knowledge Collection for NSD:


*Continued on next page*
Section I – NSD Basics, Continued

W32 NSD

When NSD is run manually on W32 using no switches, it attaches to all Domino processes, dumps all call stacks, runs Memcheck data and system information, then display an NSD prompt, where it remains attached to all Domino processes.

On Windows® 2000, quitting NSD kills all Domino processes, due to a limitation with the Operating System that does not allow for a debugger to detach from a process without exiting that process as a result. As long as the NSD prompt remains displayed, the Domino server will continue operation.

On Windows 2003 & XP, NSD displays a prompt, but you can manually detach from all processes with the command “detach”, allowing NSD to quit without affecting the Domino Server.

When run as part of Fault Recovery, NSD is no longer run from JIT interface on W32. As a result, NSD will only be invoked for crashes occurring in a Domino process (any process that makes use of the Domino API).

Common NSD switches on W32

- `nsd –detach` (detaches from all processes without killing them - XP & 2003 only) => `nsd*.log`
- `nsd –stack` (collects only call stacks, speeds execution) => `nsd*.log`
- `nsd –info` (collects only system info) => `sysinfo*.log`
- `nsd –noinfo` (collects all but info) => `nsd*.log`
- `nsd –memcheck` (collects only memcheck info) => `memcheck*.log`
- `nsd –nomemcheck` (collects all but memcheck) => `nsd*.log`
- `nsd –perf` (collects process memory usage) => `perf*.log`
- `nsd –noperf` (collects all but performance data) => `nsd*.log`
- `nsd –handles` (collects OS level handle info) => `handles*.log`
- `nsd –nohandles` (collects all but handle info) => `nsd*.log`
- `nsd –kill` (kills all notes process and associated memory) => `nsd_kill*.log`
- `nsd –monitor` (attaches and waits for exceptions) => `nsd*.log`
- `nsd –p` (runs against a specific process – call stacks only) => `nsd*.log`

At NSD prompt

Can use all of these commands plus:

- `dump` (dumps all call stacks out)
- `quit –f` (forces a quit – will bring down all Domino processes)

Continued on next page
Section I – NSD Basics, Continued

UNIX NSD
On Unix, when run manually with no switches, NSD attaches to all Domino processes,
umps data, and detached automatically (returning to a system prompt). NSD behaves
differently on Unix than W32 because the Unix debugging interface allows for NSD to
detach from all processes without causing any problems. Unix platforms include IBM
AIX®, Linux®, SUN Solaris®, and IBM zSeries®.

Common
NSD
Switches on
UNIX
• nsd –batch (runs nsd with no output to console)
• nsd –info (collects only system info) => nsd_sysinfo*.log
• nsd –noinfo (collects all but system info) => nsd_all*.log
• nsd –memcheck (collects only memcheck info) => nsd_memcheck*.log
• nsd –nomemcheck (collects all but memcheck) => nsd_all*.log
• nsd –kill (kills all notes process and associated memory) => nsd_killer*.log
• nsd –ps (lists running processes) => nsd_ps_*.log
• nsd –lsof (list of open Notes file) => nsd_lsof*.log
• nsd –nolsof (collects all but lsof) => nsd_all*.log
• nsd –user (runs nsd as specific unix user)

Note: There is no NSD prompt on Unix, since NSD does not have the detach problems
on Unix that it does on W32.

NSD on
iSeries
NSD is a different tool in ND6 on the IBM iSeries® platform from the other platforms. In
ND6, NSD output includes:

• Call stacks
• Environment Info
• Job Log of current jobs
• Status of current jobs
• Thread/Mutex info
• Notes.ini
• Last 2000 lines of Console log

Note: NSD takes no switches on iSeries, only runs for fault recovery. DumpSrvStacks is
a separate utility available for dumping call stacks. In ND7, changes have been made to
NSD on iSeries to function more closely to NSD on the other platforms. Support is in the
process of updating Knowledge Base to reflect these new capabilities.

Continued on next page
Section I – NSD Basics, Continued

NSD Major Sections

As you may or may not know, NSD output is rather verbose. In order to more easily visualize the contents of NSD, you can consider that there are four major sections within an NSD log, each major section being composed of subsections, referred to as minor sections. The four major sections within NSD are:

- **Process Information (Call Stacks)** – Process Information is composed of the list of all running processes system wide, followed by a list of all Domino specific processes. However, the most important aspect in this section is the list of the Call Stacks for all threads of each Domino Process. These call stacks provide Support with insight into the code path involved in a particular problem. This section is arguably the most important section within NSD.

- **Memcheck (Domino Memory Objects)** – The Memcheck section dumps information about Domino-specific structures. Memcheck steps through all Domino allocated memory pools (both private and shared). Memcheck output is composed of a series of minor sections which summarize memory usage and provide a list of open resources such as open database, open view, open documents, connected users, and open files. Memcheck tells Support which resources are involved in a given problem. Hence, Memcheck is a runner up for most important part of NSD.

- **System Information** – This section provides information regarding version of OS, kernel configurations, patch information, disk information, network connections, and memory usage. While this section is important under a number of different cases, this lab does not focus on information within this section.

- **Environment Information** – The section rounds out NSD output, providing the user environment, the notes.ini, and a list of the Domino executables and file located within the Domino Data directory and sub-directories. Again, this lab will not focus on this section.
Section II – Call Stacks

What is a Call Stack?

For most server or client crashes or hangs, the Call Stack section is the single most important section of NSD. But what exactly is a call stack? In brief, each thread has its own area of memory referred to as its stack region which is reserved for the execution of functions. The region of memory is used as a temporary scratch-pad to store CPU register values, arguments to each function, variables that are local to each function, and return addresses for each caller (i.e. where to resume activity).

The summary of return addresses listed in a thread’s stack region essentially provides an indication of the code path of execution for that thread at any given moment. This summary, also referred to as a stack trace summary, is constantly changing depending on thread activity. It is this summary that NSD dumps for each and every Domino thread, providing a snapshot of the code path of activity for each thread. This summary is invaluable in troubleshooting crashes and hangs.

Note: As important as call stacks are, they provide limited information. Namely, call stacks provide the current state of thread activity – not a cumulative history of thread activity. Stacks frames show current conditions, but not necessarily a history of how those conditions were reached. With complex problems, it is often necessary to augment call stack data with other forms of debug to develop a clearer picture of the problem.

Call Stacks in NSD

The level of information that NSD provides for call stacks differs for each platform. In general, platforms fall into two categories: W32 and UNIX.

W32

For the fatal thread, NSD makes 3 passes (2 passes in the original format):
• 1). Dumps complete call stack (divided into "before" and "after" frames)
• 2). Granular break down of stack frames, showing arguments, return address, basic register information
• 3). De-referenced pointer arguments for each function, meaning if a function is passed a pointer, NSD dumps out the contents of memory that the pointer references

UNIX

This includes Linux, AIX, Solaris, z/OS and i5/OS. NSD provides one pass for call stack, and currently does not break down stack frames. Limited register data is provided for certain platforms (AIX, Linux & z/OS). On AIX - arguments may show as "???", meaning the module in question was code not compiled with the necessary levels of debug to show argument information.

Continued on next page
Section II – Call Stacks, Continued

W32 Call Stacks

On W32, NSD provides 2 passes of information or the fatal thread, or 3 passes in ND7. Below are excerpts from an NSD for each pass. For non-fatal threads, only PASS ONE is provided.

**PASS ONE**

This pass provides a stack trace summary for the thread, divided into “before” and “after” frames. NSD first lists the “after” frames – these are all frames that indicate how the thread attempts to handle the fatal condition. These frames do NOT indicate the fatal condition itself. In essence, you can ignore the “after” frames.

```
############################################################
### thread 67/135: [ nSERVER:0908: 2692]
### FP=0x0ae5ec70, PC=0x6018eb23, SP=0x0ae5e2f8
### stkbase=0x0ae60000, total stksize=262144, used stksize=2424
############################################################
[ 1] 0x77f83786 ntdll.ZwWaitForSingleObject+11 (560,36ee80,0,601a7c06)
[ 2] 0x77e87837 KERNEL32.WaitForSingleObject+15 (7e4e5a0,77e8ae88,7e4ec0c,0)
@ [ 3] 0x601a7046 nnotes._OSFaultCleanup@12+342 (0,0,0,7e4ec0c)
@ [ 4] 0x601b07b1 nnotes._OSNTUnhandledExceptionFilter@4+145(7e4ec0c,7e4ec0c,6ef1ab5,7e4ec0c)
[ 5] 0x1000e596 jvm._JVM_FindSignal@84+180 (7e4ec0c,77ea18a5,7e4ec14,0)
[ 6] 0x77eae9e0 KERNEL32.CloseProfileUserMapping+161 (0,0,0,0)
```

The “before” frames are all thread activity leading up to an including the crash. The “before” frames are actually listed second, so don’t be confused. These frames are the real meat of the crash – this is the part you want to examine:

```
############################################################
### FATAL THREAD 67/135 [ nSERVER:0908: 2692]
### FP=0x0ae5ec70, PC=0x6018eb23, SP=0x0ae5e2f8
### stkbase=0x0ae60000, total stksize=262144, used stksize=2424
### EAX=0x010d088c, EBX=0x00000000, ECX=0x00ba0000, EDX=0x00ba0000
### ESI=0x0ae5e904, EDI=0x00001d34, CS=0x0000001b, SS=0x00000023
### DS=0x00000023, ES=0x00000023, FS=0x0000003b, GS=0x00000000 Flags=0x00010202
### Exception code: c0000005 (ACCESS_VIOLATION)
############################################################
@ [ 1] 0x6018eb23 nnotes._Panic@4+483 (609d0013,0,ae5eeb8,6010ee14)
@ [ 2] 0x6018e8ec nnotes._Halt@4+28 (107,0,0,0)
@ [ 3] 0x6010ee14 nnotes._MemoryInit+212 (0,0,0,0)
[ 4] 0x6010f05e nnotes._OSInit+14 (0,0,e9754f4,5a4cc160)
[ 5] 0x626412d2 nlsxbe._LsxMsgProc+112 (2,5a4c8288,5a4cc160,e953ff4)
[ 6] 0x6014f93 nnotes.DLLNode::Register+35 (5a4cc160,e967674,e980000,e93ff4)
[ 7] 0x6014f5f3 nnotes.LSISession::RegisterClassLibrary+105 (ae5f8a0,ae5f894,ae5f884,6014f59d)
[ 8] 0x6014f59d nnotes.LSSession::RegisterClassLibrary+19 (e967674,ae5f8a0,e953ff4,ae5f894)
[ 9] 0x6014f59d nnotes.LSISession::RegisterClassLibrary+141 (e967674,ae5f8a0,321,e964f8f)
[10] 0x6014f59d nnotes.LSSession::RegisterClassLibrary+141 (e967674,ae5f8a0,321,e964f8f)
[11] 0x6014f59d nnotes.LSSession::RegisterClassLibrary+19 (e967674,ae5f8a0,e953ff4,ae5f894)
[12] 0x6014f59d nnotes.LSSession::RegisterClassLibrary+141 (e967674,ae5f8a0,321,e964f8f)
[13] 0x6014f59d nnotes.LSSession::RegisterClassLibrary+19 (e967674,ae5f8a0,e953ff4,ae5f894)
[14] 0x6014f59d nnotes.LSSession::RegisterClassLibrary+141 (e967674,ae5f8a0,321,e964f8f)
```

Frames that are pre-fixed with the ‘@’ symbol mean these are functions that NSD was able to annotate using the Domino symbol files.

---

Rob Gearhart/Elliott Harden

8

HND202 - Lotusphere 2007
Section II – Call Stacks, Continued

W32 Call Stacks
(continued)  

PASS TWO

This pass provides a break down of stack frame contents, showing local variables, arguments, etc. To the right is an ASCII representation of the stack frame contents. This ASCII column can provide insight such as an error message, a database name, or user name.

Be Careful  While the ASCII section in PASS TWO is helpful, be careful not to jump to conclusions based on this information. The values you see are variables that are local each function. While these variables are important to examine, they may not be directly relevant to the root-cause of a given crash.

Be Careful  While the ASCII section in PASS TWO is helpful, be careful not to jump to conclusions based on this information. The values you see are variables that are local each function. While these variables are important to examine, they may not be directly relevant to the root-cause of a given crash.

Be Careful  While the ASCII section in PASS TWO is helpful, be careful not to jump to conclusions based on this information. The values you see are variables that are local each function. While these variables are important to examine, they may not be directly relevant to the root-cause of a given crash.

Be Careful  While the ASCII section in PASS TWO is helpful, be careful not to jump to conclusions based on this information. The values you see are variables that are local each function. While these variables are important to examine, they may not be directly relevant to the root-cause of a given crash.

Be Careful  While the ASCII section in PASS TWO is helpful, be careful not to jump to conclusions based on this information. The values you see are variables that are local each function. While these variables are important to examine, they may not be directly relevant to the root-cause of a given crash.

Be Careful  While the ASCII section in PASS TWO is helpful, be careful not to jump to conclusions based on this information. The values you see are variables that are local each function. While these variables are important to examine, they may not be directly relevant to the root-cause of a given crash.

Be Careful  While the ASCII section in PASS TWO is helpful, be careful not to jump to conclusions based on this information. The values you see are variables that are local each function. While these variables are important to examine, they may not be directly relevant to the root-cause of a given crash.

Be Careful  While the ASCII section in PASS TWO is helpful, be careful not to jump to conclusions based on this information. The values you see are variables that are local each function. While these variables are important to examine, they may not be directly relevant to the root-cause of a given crash.

Be Careful  While the ASCII section in PASS TWO is helpful, be careful not to jump to conclusions based on this information. The values you see are variables that are local each function. While these variables are important to examine, they may not be directly relevant to the root-cause of a given crash.

Be Careful  While the ASCII section in PASS TWO is helpful, be careful not to jump to conclusions based on this information. The values you see are variables that are local each function. While these variables are important to examine, they may not be directly relevant to the root-cause of a given crash.

Be Careful  While the ASCII section in PASS TWO is helpful, be careful not to jump to conclusions based on this information. The values you see are variables that are local each function. While these variables are important to examine, they may not be directly relevant to the root-cause of a given crash.

Be Careful  While the ASCII section in PASS TWO is helpful, be careful not to jump to conclusions based on this information. The values you see are variables that are local each function. While these variables are important to examine, they may not be directly relevant to the root-cause of a given crash.

Be Careful  While the ASCII section in PASS TWO is helpful, be careful not to jump to conclusions based on this information. The values you see are variables that are local each function. While these variables are important to examine, they may not be directly relevant to the root-cause of a given crash.

Be Careful  While the ASCII section in PASS TWO is helpful, be careful not to jump to conclusions based on this information. The values you see are variables that are local each function. While these variables are important to examine, they may not be directly relevant to the root-cause of a given crash.

Be Careful  While the ASCII section in PASS TWO is helpful, be careful not to jump to conclusions based on this information. The values you see are variables that are local each function. While these variables are important to examine, they may not be directly relevant to the root-cause of a given crash.
Section II – Call Stacks, Continued

UNIX Call Stacks
On Unix, NSD provides only the stack trace summary (PASS ONE). However, just as is the case with W32, the upper part of the stack shows thread activity that attempts to handle the fatal exception, and does not indicate the exception itself. Look at portion of the call stack below the “fatal”, “raise.raise”, “signal handler”, “abort” or “terminate” line. The exact syntax for this line will depend on the version of UNIX, as well as the nature of the problem itself. In addition, C++ function names are mangled for all UNIX platforms except zSeries (OS390), so they can be difficult to read. Below are some examples of call stacks for each platform.

Note: For iSeries NSDs, call stacks are read from bottom (most recent function) to top (previous callers), whereas all other platforms are read the opposite, from top (most recent function) to bottom (previous callers).

Function Mangling
Function mangling is an artifact of the compile process for software. In order to uniquely identify one function from another, the compiler generates a function decoration, or mangles the function, which includes the function name, class (if any), and argument types. As a result, it takes a bit of work to extract the function name from the function decoration.

Within NSD, the syntax for function decorations is very similar for AIX, Linux, and iSeries. Solaris has a different decoration syntax; zSeries function names are not mangles within NSD output.

Continued on next page
Section II – Call Stacks, Continued

### AIX Call Stack

```c
## thread 6/15 :: adminp pid=70336, k-id=275121, pthr-id=1286
## stack          :: k-state=wait, stk max-size=262144, cur-size=9484
ptrgl.$PTRGL() at 0xd01d7a10
raise.nsleep(??, ??) at 0xd01e6490
raise.nsleep(??, ??) at 0xd01e6490
sleep(??) at 0xd02515b0
OSRunExternalScript(??) at 0xd3e2bcf8
OSFaultCleanup(??, ??, ??) at 0xd3e2cf54
fatal_error(??, ??, ??) at 0xd4d3f5cc
pth_signal.pthread_kill(??, ??) at 0xd0199cf0
raise.raise(??) at 0xd01e688c
Panic(??) at 0xd3c621b8
LockHandle(??, ??, ??) at 0xd3c698dc
OSLockObject(??) at 0xd3c6a358
ItemLookupByName(??, ??, ??, ??, ??) at 0xd3e928c4
NSFItemLookupByName(??, ??, ??, ??, ??) at 0xd3e92d6c
NSFItemInfo(??, ??, ??, ??, ??, ??, ??) at 0xd3e9250c
AdminpCompileResponseStatus(0x3b003b, 0x0, 0x3038d250, 0x3038d150, 0x100a2340) at 0x10006324
CompileResponseStatus__12AdminRequestFUsPcT2PCc(??, ??, ??, ??, ??) at 0x10054cb0
DoProcessRequest__21CreateIMAPDelegations(??) at 0x1005372c
AdminProcessNewRequest(??, ??, ??, ??, ??, ??) at 0x1005372c
EntryThread(??) at 0x10002f80
ThreadWrapper(??) at 0xd3c5f4ac
pth_pthread._pthread_body(??) at 0xd018a5a4
```

### Linux Call Stack

```c
---- Thread 12759
0x42174771: __nanosleep + 0x11 (1, 41f9ac54, 3150, 1, 4bc08608, 0) + 354
0x406a2421: OSRunExternalScript + 0x15d (0, 41f9ac54, 4bc08a1c, 4bc08a1c, 61660000, ffff) + 1c8
0x406a14e2: OSFaultCleanup + 0x4b2 (0, 0, 0, 8141f6c, 6, 4bc08cc8) + 20c
0x494af892: panicSignalHandler + 0xea (6, 4bc09088, 0, fffffe64, 20, 0) + 110
0x494ee80f: sysUnwindSignalCatchFrame + 0x77 (494ee844, 6, 4bc08cc8) + 20c
0x494ee8a1: sysSignalCatchHandler + 0x5d (6, 4bc09088, 0, fffffe64, 20, 0) + 110
0x494ef12c: userSignalHandler + 0x68 (6, 4bc08cc8, 0, fffffe64, 20, 0) + 110
0x494ef0ba: intrDispatch + 0xba (6, 4bc08cc8, 4bc0865c, 4bc0865c) + 54
0x494ef31c: intrDispatchMD + 0x60 (6, 4bc08cc8, 4bc0865c, 4bc0865c) + 54
0x42749c15: default_terminate + 0x15 (428e4d8c)
0x42747c15: default_terminate + 0x15 (428e4d8c)
0x42747c15: default_terminate + 0x15 (428e4d8c)
0x42747c15: default_terminate + 0x15 (428e4d8c)
0x42747c15: default_terminate + 0x15 (428e4d8c)
```

--- 

Continued on next page
### iSeries Call Stack

<table>
<thead>
<tr>
<th>PID/TID</th>
<th>Function/Class Names</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x34</td>
<td>QLERTTH QLESPI</td>
</tr>
<tr>
<td>18</td>
<td>THREAD LIBNOTES</td>
</tr>
<tr>
<td>9</td>
<td>HTTHREAD LIBHTTPSTA</td>
</tr>
<tr>
<td>9</td>
<td>HTWRKTHR</td>
</tr>
<tr>
<td>75</td>
<td></td>
</tr>
<tr>
<td>150</td>
<td></td>
</tr>
<tr>
<td>330</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>256</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>CMDHAND</td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>115</td>
<td>CMDHANDB</td>
</tr>
<tr>
<td>9</td>
<td>OPIMGHD</td>
</tr>
<tr>
<td>194</td>
<td>CMD</td>
</tr>
<tr>
<td>2</td>
<td>NDB</td>
</tr>
<tr>
<td>1</td>
<td>NSFSEMM LINES</td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>DBLOCK</td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>OSPANIC</td>
</tr>
<tr>
<td>33</td>
<td>BREAK</td>
</tr>
<tr>
<td>1</td>
<td>CLEANUP</td>
</tr>
</tbody>
</table>

Read stack from here up (on iSeries)

--- called from signal handler with signal 11 (SIGSEGV)

--- called from signal handler with signal 12

--- called from signal handler with signal 13

### Solaris Call Stack

<table>
<thead>
<tr>
<th>PID</th>
<th>Function/Class Names</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x34</td>
<td>QLERTTH QLESPI</td>
</tr>
<tr>
<td>18</td>
<td>THREAD LIBNOTES</td>
</tr>
<tr>
<td>9</td>
<td>HTTHREAD LIBHTTPSTA</td>
</tr>
<tr>
<td>9</td>
<td>HTWRKTHR</td>
</tr>
<tr>
<td>75</td>
<td></td>
</tr>
<tr>
<td>150</td>
<td></td>
</tr>
<tr>
<td>330</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>256</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>CMDHAND</td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>115</td>
<td>CMDHANDB</td>
</tr>
<tr>
<td>9</td>
<td>OPIMGHD</td>
</tr>
<tr>
<td>194</td>
<td>CMD</td>
</tr>
<tr>
<td>2</td>
<td>NDB</td>
</tr>
<tr>
<td>1</td>
<td>NSFSEMM LINES</td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>DBLOCK</td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>OSPANIC</td>
</tr>
<tr>
<td>33</td>
<td>BREAK</td>
</tr>
<tr>
<td>1</td>
<td>CLEANUP</td>
</tr>
</tbody>
</table>

Read stack from here down

--- called from signal handler with signal 11 (SIGSEGV)

--- called from signal handler with signal 12

--- called from signal handler with signal 13

--- called from signal handler with signal 14

--- called from signal handler with signal 15

--- called from signal handler with signal 16

--- called from signal handler with signal 17

--- called from signal handler with signal 18

--- called from signal handler with signal 19

--- called from signal handler with signal 20

--- called from signal handler with signal 21

--- called from signal handler with signal 22

--- called from signal handler with signal 23

--- called from signal handler with signal 24

--- called from signal handler with signal 25

--- called from signal handler with signal 26

--- called from signal handler with signal 27

--- called from signal handler with signal 28

--- called from signal handler with signal 29

--- called from signal handler with signal 30

--- called from signal handler with signal 31

--- called from signal handler with signal 32

--- called from signal handler with signal 33

--- called from signal handler with signal 34

--- called from signal handler with signal 35

--- called from signal handler with signal 36

--- called from signal handler with signal 37

--- called from signal handler with signal 38

--- called from signal handler with signal 39

--- called from signal handler with signal 40

--- called from signal handler with signal 41

--- called from signal handler with signal 42

--- called from signal handler with signal 43

--- called from signal handler with signal 44

--- called from signal handler with signal 45

--- called from signal handler with signal 46

--- called from signal handler with signal 47

--- called from signal handler with signal 48

--- called from signal handler with signal 49

--- called from signal handler with signal 50

--- called from signal handler with signal 51

--- called from signal handler with signal 52

--- called from signal handler with signal 53

--- called from signal handler with signal 54

--- called from signal handler with signal 55

--- called from signal handler with signal 56

--- called from signal handler with signal 57

--- called from signal handler with signal 58

--- called from signal handler with signal 59

--- called from signal handler with signal 60

--- called from signal handler with signal 61

--- called from signal handler with signal 62

--- called from signal handler with signal 63

--- called from signal handler with signal 64

--- called from signal handler with signal 65

--- called from signal handler with signal 66

--- called from signal handler with signal 67

--- called from signal handler with signal 68

--- called from signal handler with signal 69

--- called from signal handler with signal 70

--- called from signal handler with signal 71

--- called from signal handler with signal 72

--- called from signal handler with signal 73

--- called from signal handler with signal 74

--- called from signal handler with signal 75

--- called from signal handler with signal 76

--- called from signal handler with signal 77

--- called from signal handler with signal 78

--- called from signal handler with signal 79

--- called from signal handler with signal 80

--- called from signal handler with signal 81

--- called from signal handler with signal 82

--- called from signal handler with signal 83

--- called from signal handler with signal 84

--- called from signal handler with signal 85

--- called from signal handler with signal 86

--- called from signal handler with signal 87

--- called from signal handler with signal 88

--- called from signal handler with signal 89

--- called from signal handler with signal 90

--- called from signal handler with signal 91

--- called from signal handler with signal 92

--- called from signal handler with signal 93

--- called from signal handler with signal 94

--- called from signal handler with signal 95

--- called from signal handler with signal 96

--- called from signal handler with signal 97

--- called from signal handler with signal 98

--- called from signal handler with signal 99

--- called from signal handler with signal 100

--- called from signal handler with signal 101

--- called from signal handler with signal 102

--- called from signal handler with signal 103

--- called from signal handler with signal 104

--- called from signal handler with signal 105

--- called from signal handler with signal 106

--- called from signal handler with signal 107

--- called from signal handler with signal 108

--- called from signal handler with signal 109

--- called from signal handler with signal 110

--- called from signal handler with signal 111

--- called from signal handler with signal 112

--- called from signal handler with signal 113

--- called from signal handler with signal 114

--- called from signal handler with signal 115

--- called from signal handler with signal 116

--- called from signal handler with signal 117

--- called from signal handler with signal 118

--- called from signal handler with signal 119

--- called from signal handler with signal 120
Section II – Call Stacks, Continued

zSeries Call Stack

# thread 1/4 :: update pid=340, k-id=0x12850600, pthr-id=
# stack :: k-state=activ, stk max-size=0, cur-size=0

sleep() at 0x12370204
OSRunExternalScript() at 0x12a3255c
OSFaultCleanup() at 0x12a35226
 fatal_error() at 0x12a20eb8
__zerros() at 0x1238d164
() at 0x9e67426
CGtrPosWork::ReadNext(unsigned char)() at 0x9e67426
CGtrPosWork::InsertDoc (CGtrPosSh)() at 0x1650036a
CGtrPosBrokerNormal::Externalize (KEY_R)() at 0x164ae9ec
gtr_MergePatt() at 0x1648738a
GTR_mergeIndex() at 0x16428f36
cGTRio::Merge()() at 0x163fb848
FTGMergeGTRIndexes(FTG_CTX*,int)() at 0x163f2210
FTGIndex() at 0x163df398
FTCallIndex() at 0x141b3910

Class Name (not mangled)
Function Name (not mangled)
Section III: Memcheck – Shared Memory

Overview

Shared Memory is one of the most important aspects of troubleshooting Domino issues. The Memcheck section of NSD dumps out verbose information regarding Shared Memory Usage and Shared Memory structures. We touch on the more important aspects of Memcheck of which you as an administrator should be aware.

<table>
<thead>
<tr>
<th>Lesson</th>
<th>See Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lesson I - Summary of Shared Pools</td>
<td>15</td>
</tr>
<tr>
<td>Lesson II - Top 10 Shared Memory Block Usage</td>
<td>17</td>
</tr>
<tr>
<td>Lesson III - Shared OS Fields (MM/OS Structure)</td>
<td>19</td>
</tr>
<tr>
<td>Lesson IV - Open Databases</td>
<td>20</td>
</tr>
<tr>
<td>Lesson V - Open Documents</td>
<td>22</td>
</tr>
<tr>
<td>Lesson IV – Open Views</td>
<td>25</td>
</tr>
</tbody>
</table>

Using Memcheck Output

Realize that you will never be looking at NSD output in a vacuum. Usually, you will be dealing with NSD in the context of an issue, such as a crash or performance problem. You should use Memcheck in conjunction with other NSD components (namely call stacks) as well as other Domino diagnostics (such as console or semaphore output) or OS diagnostics (such as PerfMon on W32, or PerfPMR on AIX).

Note about Memory Usage

The summarization of shared (and private) memory usage in Memcheck includes memory allocated ONLY by Domino. Memory usage by LotusScript, as well as other components such as Java and other third party components is not included in this summary. In order to evaluate total memory usage for a process, OS diagnostics are typically needed.
Lesson I – Summary of Shared Pools

How to Find This Section

• Search on KEYWORD "memcheck" until you come to the beginning of the memcheck section.
• Search on KEYWORDS "Shared Memory: (ND6) to find the total shared memory usage across all shared Domino Pools
• Search on KEYWORDS "Shared Memory Stats" (ND7) to find the total shared memory usage across all shared Domino Pools

Original Format (ND6.5.4 and earlier)

<table>
<thead>
<tr>
<th>TYPE</th>
<th>Count</th>
<th>SIZE</th>
<th>ALLOC</th>
<th>FREE</th>
<th>FRAG</th>
<th>OVERHEAD</th>
<th>%used</th>
<th>%free</th>
</tr>
</thead>
<tbody>
<tr>
<td>S-DPOOL:</td>
<td>294</td>
<td>159500000000</td>
<td>952976300</td>
<td>641584608</td>
<td>0</td>
<td>753136</td>
<td>59%</td>
<td>40%</td>
</tr>
<tr>
<td>Overall:</td>
<td>294</td>
<td>159500000000</td>
<td>952976300</td>
<td>641584608</td>
<td>0</td>
<td>753136</td>
<td>59%</td>
<td>40%</td>
</tr>
</tbody>
</table>

Newer Format (ND6.5.5 and later)

<@@ ------ Notes Memory Analyzer (memcheck) -> Shared Memory Stats (Time 17:45:55) ------ @@>

<table>
<thead>
<tr>
<th>TYPE</th>
<th>Count</th>
<th>SIZE</th>
<th>ALLOC</th>
<th>FREE</th>
<th>FRAG</th>
<th>OVERHEAD</th>
<th>%used</th>
<th>%free</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static-DPOOL:</td>
<td>35</td>
<td>125829120</td>
<td>116479928</td>
<td>9334512</td>
<td>0</td>
<td>19408</td>
<td>92%</td>
<td>7%</td>
</tr>
<tr>
<td>Overall</td>
<td>35</td>
<td>125829120</td>
<td>116479928</td>
<td>9334512</td>
<td>0</td>
<td>19408</td>
<td>92%</td>
<td>7%</td>
</tr>
</tbody>
</table>

Field Label Description

<table>
<thead>
<tr>
<th>Field Label</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE</td>
<td>Indicates the various types of pools that are allocated. In almost all cases, all pools will be labeled as “S-DPOOL”, or “Static DPOOL”, and will match the figures for “Overall”</td>
</tr>
<tr>
<td>SIZE</td>
<td>Indicates the total amount of memory allocated by the Domino Memory Manager, not all of which may be in use. This total should be no higher than about 1.0 GB to 1.2 GB (see note below).</td>
</tr>
<tr>
<td>ALLOC</td>
<td>Indicates the amount of the “SIZE” that is actually in use (or sub-allocated) at any given time.</td>
</tr>
<tr>
<td>%used</td>
<td>Indicates the percentage used of memory (ALLOC/SIZE). It’s a good thing to have percentages in the 90% range, not a bad thing. The higher the %used, the better – we want to be making good use of what we have allocated.</td>
</tr>
</tbody>
</table>

How much shared memory?

Every server’s configuration and usage are different; therefore the amount of memory usage for every server will also be different. However, as a rule of thumb, you should usually see around 1.0 to 1.2 GB of shared memory usage. A majority of this memory should be in the form of the UBM (0x82cd), at around 750 MB or less. While a higher number for shared memory usage may not indicate a server defect, it should be investigated to see if any configuration changes need to be made. Certain platforms such as Solaris can use more shared memory without detriment, upwards of 1.5 GB – 1.6 GB.
Lesson I – Summary of Shared Pools, Continued

How to Use This Section
Use this section of Memcheck when dealing with high shared memory usage, or some problem resulting from high shared memory usage. You may also use this section if you simply want to establish the amount of shared memory allocated by the Domino Server at any given time.

Upon examination of the Summary of Shared DPOOLs:

<table>
<thead>
<tr>
<th>IF…</th>
<th>THEN…</th>
<th>AND You Should…</th>
</tr>
</thead>
<tbody>
<tr>
<td>• you find lots of DPOOLS (more than 500 depending on platform) – OR -&lt;br&gt;• you find high total shared memory usage (more than about 1.2 GB)</td>
<td>you may be dealing with high memory usage or a leak</td>
<td>check the Top 10 Shared Memory Block Usage. While this section gives you the total amount of memory allocated for all DPOOLS, it does NOT provide a break down memory usage per block (for that, see the Top 10 section). Call into Support for assistance examining files</td>
</tr>
<tr>
<td>• you find a low %used figure (for instance, below 85%)</td>
<td>you may be dealing with poor DPOOL utilization</td>
<td>collect memory dumps. You will need memory dumps to determine if fragmentation is occurring for sure. Call into Support for assistance examining files</td>
</tr>
</tbody>
</table>

Note: In the excerpt above, the server has 294 Shared DPOOLS allocated, with overall shared memory usage over 1.5 GB. This figure is above the threshold of 1.2 GB and should be investigated. The Top 10 Shared Memory Block Usage is usually the next place to go. Needless to say, you want to initiate a call with Support.
Lesson II – Top 10 Shared Memory Block Usage

How to Find This Section
- KEYWORD "Top 10 Shared Memory"

Original Format (ND6.5.4 and earlier)

<table>
<thead>
<tr>
<th>Type</th>
<th>TotalSize</th>
<th>Handles</th>
<th>Type</th>
<th>Handles</th>
<th>TotalSize</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x82cd</td>
<td>78314944</td>
<td>199</td>
<td>0x841c</td>
<td>2673</td>
<td>3803910</td>
</tr>
<tr>
<td>0x82cc</td>
<td>14519040</td>
<td>199</td>
<td>0x841b</td>
<td>2673</td>
<td>107708</td>
</tr>
<tr>
<td>0x8219</td>
<td>5250452</td>
<td>491</td>
<td>0x8405</td>
<td>663</td>
<td>1578194</td>
</tr>
<tr>
<td>0x8252</td>
<td>5242880</td>
<td>5</td>
<td>0x8001</td>
<td>609</td>
<td>824586</td>
</tr>
<tr>
<td>0x841c</td>
<td>3803910</td>
<td>2673</td>
<td>0x8219</td>
<td>491</td>
<td>5250452</td>
</tr>
<tr>
<td>0x834a</td>
<td>3670016</td>
<td>4</td>
<td>0x82d2</td>
<td>424</td>
<td>1054622</td>
</tr>
<tr>
<td>0x82df</td>
<td>2097152</td>
<td>1</td>
<td>0x8439</td>
<td>318</td>
<td>450932</td>
</tr>
<tr>
<td>0x8a05</td>
<td>1578194</td>
<td>663</td>
<td>0x8231</td>
<td>278</td>
<td>127260</td>
</tr>
<tr>
<td>0x8f56</td>
<td>1111902</td>
<td>17</td>
<td>0x8232</td>
<td>278</td>
<td>2506</td>
</tr>
</tbody>
</table>

Newer Format (ND6.5.5 and later)

<table>
<thead>
<tr>
<th>Type</th>
<th>TotalSize</th>
<th>Handles</th>
<th>Typename</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x82cd</td>
<td>637673472</td>
<td>162</td>
<td>BLK_UMBUFFER</td>
</tr>
<tr>
<td>0x8252</td>
<td>20971520</td>
<td>20</td>
<td>BLK_NSF_POOL</td>
</tr>
<tr>
<td>0x834a</td>
<td>18350080</td>
<td>18</td>
<td>BLK_GB_CACHE</td>
</tr>
<tr>
<td>0x82cc</td>
<td>10511340</td>
<td>161</td>
<td>BLK_UBMBCB</td>
</tr>
<tr>
<td>0x8a03</td>
<td>9810466</td>
<td>160</td>
<td>BLK_OPENED_NOTE</td>
</tr>
<tr>
<td>0x8a05</td>
<td>6760070</td>
<td>1604</td>
<td>BLK_NETBUFFER</td>
</tr>
<tr>
<td>0x8311</td>
<td>5242880</td>
<td>5</td>
<td>BLK_NIF_POOL</td>
</tr>
<tr>
<td>0x890b</td>
<td>4578420</td>
<td>70</td>
<td>BLK_EXECPOOL</td>
</tr>
<tr>
<td>0x8a05</td>
<td>2460000</td>
<td>1</td>
<td>BLK_NET_SESSION_TABLE</td>
</tr>
<tr>
<td>0x8a01</td>
<td>2289210</td>
<td>35</td>
<td>BLK_NETPOOL</td>
</tr>
</tbody>
</table>

Top 10 Shared Memory Block Usage shows a list of the top 10 highest users of memory blocks by total size and by number of blocks. Note: the above excerpt for the Newer Format has been truncated for clarity.

<table>
<thead>
<tr>
<th>Field Label</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>The hexadecimal block type designation used by the component that allocated the block</td>
</tr>
<tr>
<td>TotalSize</td>
<td>Amount of shared memory allocated across all blocks of a given type</td>
</tr>
<tr>
<td>Handles</td>
<td>Number of blocks allocated for a given type (each block uses one handle)</td>
</tr>
</tbody>
</table>

Continued on next page
Lesson II – Top 10 Shared Memory Block Usage, Continued

How to Use This Section
This section augments the Shared Pool Summary by breaking down memory usage by block type. This output ONLY shows block usage, not pool usage. Hence, if a pool is not well utilized, you will not see the problem here. Under those cases, you should collect a memory dump. You should come here:

• if you suspect that a shared block is leaking
• if you get "Out of Shared Handles" errors
• if you want to see shared block usage (i.e. who is using what)

Look for either a large amount of overall memory coming from one block type (exhausting the user address space) or a large number of one block type (causing “Out of shared handle” error messages).

Note: From the Top 10 excerpt above, you see the largest amount of shared memory comes from block 0x82cd (BLK_UBMBUFFER), which will typically be 500 MB to 750 MB. This usage is from the UBM, and is perfectly normal and expected. Memory usage from any other single block type should usually be an order of magnitude lower than UBM (for instance 75 MB)
Lesson III – Shared OS Fields (MM/OS Structure)

How to Find This Section

- KEYWORD "Shared OS Fields" (Original Format - ND6.5.4 and earlier)
- KEYWORD “MM/OS Structure Information” (Newer Format - ND6.5.5 and later)

Original Format (ND6.5.4 and earlier)

----- Shared OS Fields -------

Start Time = 03/23/2004 19:59:40
Crash Time = 03/23/2004 20:18:01
Error Message = PANIC: Removing Peer that was never added
SharedDPoolSize = 8388608
FaultRecovery = 0x00010013
caused Static Hang to be set
ConfigFileSem = (  SEM:#0:0x010d) n=0, wcnt=-1, Users=-1, Owner=[]
FDSem = ( RWSEM:#12:0x410f) rdcnt=-1, refcnt=0 Writer=[] n=12, wcnt=-1

Newer Format (ND6.5.5 and later)

<@@ ------ Notes Memory Analyzer (memcheck) -> MM/OS Structure Information (Time 13:15:45) ------ @@>

Start Time = 12/13/2005 01:15:02 PM
Crash Time = 12/13/2005 01:15:32 PM
Error Message = PANIC: LookupHandle: handle out of range
SharedDPoolSize = 4194304
FaultRecovery = 0x00010013
Cleanup Script Timeout= 300
Crash Limits = 3 crashes in 5 minutes
StaticHang = [   nhttp: 2752:   10]/[   nhttp: 2752: 3500] (0xac0/0xa/0xadac)
ConfigFileSem = (  SEM:#0:0x010d) n=0, wcnt=-1, Users=-1, Owner=[0:0]
FDSem = ( RWSEM:#11:0x410f) rdcnt=-1, refcnt=0 Writer=[0:0], n=11, wcnt=-1

How to Use This Section

This section gives you information about server start time, crash time, and any PANIC messages if present. It also provides the thread ID (both physical and virtual) that crashed the server, with the text “Thread” in ND6, “StaticHang” in ND7.

In theory, you can consult this section when you have multiple fatal threads or panics to determine which thread crashed first (*). Use this section if you want to establish specific crash information, although this information is available through other means, such as the NSD time and console output.

(*)CAUTION

For ND6.5.4 (and earlier) on W32, the ‘Thread’ field actually reflects the last crashing thread ID, in the case where there are two or more fatal threads. This issue has been addressed as of ND6.5.5. For a workaround, you may determine the first fatal thread by examine the OS Process Table, near the top of the NSD, and looking for the process that invoked the NSD process. This will be the process that caused the server to crash.
Lesson IV – Open Databases

How to Find This Section
• KEYWORD "Open Databases"

<@@ ------ Notes Memory Analyzer (memcheck) -> Open Databases (Time 11:45:21) ------ @@>

D:\Lotus\Domino\Data\HR\projnav.nsf
    Version    = 41.0
    ReplicaID  = 86256fe:0019c2ad
    bContQueue = NSFPool [120: 7236]
    FDGHandle = 0xf01c0928, RefCnt = 3, Dirty = N
    DBSem     = (FRWSEM:0x0244) state=0, waiters=0, refcnt=0, nlrdrs=0 Writer=()
    SemContQueue ( RWSEM:#0:0x029d) rdcnt=-1, refcnt=0 Writer=() n=0, wcnt=-1, Users=-1, Owner=()
    By:  nSERVER:0768: 132 DBH= 63984, User=CN=Rhonda Smith/O=ACME/C=US
    By: [ nSERVER:0768: 132] DBH= 64127, User=CN=Rhonda Smith/O=ACME/C=US
    By: [ nSERVER:0768: 132] DBH= 63006, User=CN=Rhonda Smith/O=ACME/C=US

D:\Lotus\Domino\Data\finance\Finance.nsf
    Version    = 41.0
    ReplicaID  = 86256c67:00664fb0
    bContQueue = NSFPool [1935: 58788]
    FDGHandle = 0xf01c2a98, RefCnt = 1, Dirty = N
    DBSem     = (FRWSEM:0x0244) state=0, waiters=0, refcnt=0, nlrdrs=0 Writer=()
    SemContQueue ( RWSEM:#0:0x029d) rdcnt=-1, refcnt=0 Writer=() n=0, wcnt=-1, Users=-1, Owner=()
    By:  nhttp:0fe4:  53 DBH= 63808, User=Anonymous

D:\Lotus\Domino\Data\manufacturing\Suppliers.nsf
    Version    = 43.0
    ReplicaID  = 86256df1:006df2839
    bContQueue = NSFPool [2014: 52900]
    FDGHandle = 0xf01c0cf6, RefCnt = 1, Dirty = Y
    DBSem     = (FRWSEM:0x0244) state=0, waiters=0, refcnt=0, nlrdrs=0 Writer=()
    SemContQueue ( RWSEM:#0:0x029d) rdcnt=-1, refcnt=0 Writer=() n=0, wcnt=-1, Users=-1, Owner=()
    By: [ nhttp:0fe4:  34] DBH= 63883, User=CN=Mary Peterson/O=ACME/C=US

<table>
<thead>
<tr>
<th>Field Label</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Database Name)</td>
<td>Name of the database. When an absolute path is given, the database has been opened locally. If the Database name contains “!CN=Server/O=ACME...”, it means the database has been opened via the network, including local databases that are opened using the server name</td>
</tr>
<tr>
<td>Version</td>
<td>Version of the database’s ODS</td>
</tr>
<tr>
<td>Replica ID</td>
<td>Self-explanatory</td>
</tr>
<tr>
<td>By: [ proc:PID:VTID]</td>
<td>Indicates the process name, process ID, and virtual thread ID of the server thread that has opened the database</td>
</tr>
<tr>
<td>DBH</td>
<td>Database handle for each instance opened (can be correlated to Open Documents)</td>
</tr>
<tr>
<td>User</td>
<td>Name of the user that has the database open. Each user can have multiple instances of a database opened (multiple DBHs)</td>
</tr>
</tbody>
</table>

Continued on next page
Lesson IV – Open Databases, Continued

How to Use This Section

The Open Databases section is one of the more important parts of Memcheck. This section gives you an indication as to what databases are opened, and what users are accessing them. Use this section to establish the total number of open databases (you must estimate this count manually), or need specific information about a database. The main information you will find helpful is:

- Database Name
- ODS Version
- Replica ID
- List of Users

While this section is valuable, you will likely find that the Resource Usage Summary is better equipped to provide immediate answers as to what databases, views, or documents a crashing or hung thread is accessing. However, this section gives more verbose information regarding database information.
Lesson V – Open Documents

How to Find This Section

- KEYWORD "Open Documents"

Original Format (ND6.5.4 and earlier)

-------- Open Documents --------

Original Format (ND6.5.4 and earlier)

Newer Format (ND6.5.5 and later)

Newer Format (ND6.5.5 and later)

Continued on next page
### Field Label Description

<table>
<thead>
<tr>
<th>Field Label</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBH</td>
<td>Database handle for each instance opened (can be correlated to Open Databases to determine database name)</td>
</tr>
<tr>
<td>NOTEID</td>
<td>NOTEID in decimal for the opened Note</td>
</tr>
<tr>
<td>CLASS</td>
<td>Class of the note – this indicates if a note is a form note, view note, agent note, or document note (see table below)</td>
</tr>
<tr>
<td>IsProf (Newer Format)</td>
<td>Indicates if the document is a profile document</td>
</tr>
<tr>
<td>Pools (Newer Format)</td>
<td>Indicates how many POOLs the document is spread across in memory</td>
</tr>
<tr>
<td>Size (Newer Format)</td>
<td>Indicates the size of the document meta data held in the above mentioned POOLs</td>
</tr>
<tr>
<td>Items (Newer Format)</td>
<td>Indicates how many items/fields the document contains</td>
</tr>
<tr>
<td>Database (Newer Format)</td>
<td>Indicates which database the document belongs to</td>
</tr>
<tr>
<td>Opened By (Newer Format)</td>
<td>Indicates which user opened the document</td>
</tr>
</tbody>
</table>

### Note Class Description

<table>
<thead>
<tr>
<th>Note Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0001</td>
<td>Data Note (document)</td>
</tr>
<tr>
<td>0x0004</td>
<td>Form Note</td>
</tr>
<tr>
<td>0x0008</td>
<td>View note</td>
</tr>
<tr>
<td>0x0040</td>
<td>ACL Note</td>
</tr>
<tr>
<td>0x0200</td>
<td>Agent Note</td>
</tr>
<tr>
<td>0x0800</td>
<td>Replication Formula Note</td>
</tr>
</tbody>
</table>

*Continued on next page*
Lesson V – Open Documents, Continued

How to use this section
You will probably find the Open Documents section one of the more helpful sections of NSD. This section shows the list of all open “notes” in memory on the server (or client), regardless of the type of note (design note, or data note, etc). You can examine this section to determine NoteID information, or establish which database a document belongs to. You are most interested in:

- DBH – database handle
- NOTEID – noteID
- CLASS – note class (e.g. NOTE_CLASS_DOCUMENT)
- FLAGS – note flags (e.g. NOTE_FLAG_UNREAD)
- Pools – how many POOLs the document is spread across in memory
- Items – how many items/fields the document contains
- Size – the size of the document meta data held in the above mentioned POOLs
- Database – which database the document belongs to
- Opened By – which user opened the document

The note's "CLASS" will tell you if a note is a document, view note, agent, etc. This information is also held in the Resource Usage Summary, and is better organized there. However, the placement of the Open Documents section within Memcheck indicates whether the notes are opened in private or shared memory, which is information not reflected in Resource Usage Summary.

Multiple Open Document Sections
Within NSD, you may encounter multiple “Open Document” sections, due to the fact that Memcheck dumps out different types of open documents as separate lists, for instance, notes opened locally, notes opened remotely, or new notes. In addition to these different note types, there may also be documents opened in shared or private memory depending on the needs of the component that opened the document. Therefore, you may potentially see as many as three open document lists in shared memory and three open documents lists under each process. It is also quite possible that you will see no open document lists at all (since no notes may be open at the time that NSD is run).

About Fast Note Opens
Whenever a server opens a note on behalf of a client, the server performs what is called a “fast note open”. In essence, the server opens the note just long enough to pass it back to the client, and then closes the note. So while a client may have a document open for long periods of time, the server keeps the note open for only a fraction of a second. Hence, open notes that are listed within a server NSD are typically in fact not opened on behalf of clients, but rather are notes that are open for the servers own use (such as during the running of an agent or mail delivery).
Lesson VI – Open Views

How to Find This Section

- KEYWORD “NIF Collections”
- KEYWORD “NIF Collection Users”

<@@ ------- Notes Memory Analyzer (memcheck) -> NIF Collections (Time 12:48:35) ------- @@>

CollectionVB ViewNoteID UNID OBJID RefCnt Flags Options Corrupt Deleted Temp NS Entries ViewTitle
------------ ------------- -------- ------ ------ ------ -------- ------- ------- ---- --- -------------- --------------
[ 0020e005] 1518 1356a8 358710 1 0x0000 00000008 NO NO NO NO 0 MyNotices

CIDB = [ 0253cc05]
CollSem (FRMSEM:0x030b) state=0, waiters=0, refcnt=0, nlrdrs=0 Writer={ : 0000}
  NumCollations = 2
  bCollationBlocks = [ 001e72e5]
  bCollation[0] = [ 00170005]
  bCollation[1] = [ 001a2205]
CollIndex = [ 00012a09]
Collation 0:BufferSize 26, Items 1, Flags 0
  0: Ascending, by KEY, "StartDateTime", summary# 2
CollIndex = [ 00012c09]
Collation 1:BufferSize 26, Items 1, Flags 0
  0: Descending, by KEY, "StartDateTime", summary# 2
ResponseIndex [ 0010e4b6]
NoteIDIndex [ 0010e385]
UNIDIndex [ 0010e5e7]

<@@ ------- Notes Memory Analyzer (memcheck) -> NIF Collection Users (hash) (Time 12:48:33) ------- @@>

CollUserVB ... CollectionVB Remote OFlags ViewNoteID Data HDB/Full View HDB/Full ... Open By
------------ ... ------------- ---------- -------------- -------------- ... -------------- ...
[ 00239805] ... [ 0023d005] NO 0x0082 786 1219/1874 1219/1874 ... [ nserver: 09d8: 04ca]
CurrentCollation = 0
[ 0013a805] ... [ 00136005] NO 0x00c2 11122 886/785 886/785 ... [ nserver: 09d8: 0266]
CurrentCollation = 0
[ 0028d805] ... [ 0028e005] NO 0x00c2 1518 551/1432 551/1432 ... [ nserver: 09d8: 03b0]
CurrentCollation = 0

Note: The string “....” indicates where output has been removed for clarity.

Continued on next page
Lesson VI – Open Views, Continued

**How to Use This Section**

*NIF Collections* provides you information about all collections opened server-wide. Use this section when you want to get an idea of the total number of open views, or if you want detailed information about the view, such as number of collations.

*NIF Collection Users* provides you information about the *users* of the views opened on the server (such as which thread has a view open, what collation it is using, etc).

For a summary of information about views being used by a specific thread, it is easier to refer to *Resource Usage Summary*, but if you need to take a deeper dive, such as investigating which threads are waiting on a semaphore for a given view, you will refer to these two sections.

The main reason to use the otherwise rather cryptic “CollectionVB” value is to cleanly correlate *NIF Collection User* information back to the *NIF Collections* information. Sometimes you can use ViewNoteID to do this, but since multiple view notes can have the same NoteID in different databases, this is not a definitive match, where as CollectionVB is definitive (since it is an in-memory location for collection info).

Below is a short description for the more important fields of information that you will routinely investigate. These fields are bolded in the excerpt on the previous page.

<table>
<thead>
<tr>
<th>Field Label</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CollectionVB <em>(Collections &amp; Collection Users)</em></td>
<td>In-memory location of collection info – good for correlating data between <em>Collections and Collection Users</em></td>
</tr>
<tr>
<td>ViewNoteID <em>(Collections &amp; Collection Users)</em></td>
<td>NoteID for view note design – sometimes – good for correlating data between <em>Collections and Collection Users</em></td>
</tr>
<tr>
<td>RefCnt <em>(NIF Collections)</em></td>
<td>Number of users that have the view open</td>
</tr>
<tr>
<td>ViewTitle <em>(NIF Collections)</em></td>
<td>Name of the View/Collection</td>
</tr>
<tr>
<td>CollSem <em>(NIF Collections)</em></td>
<td>Info about the Collection Semaphore. If contention exists on a view, contains reader and waiter information (same as semdebug)</td>
</tr>
<tr>
<td>NumCollations <em>(NIF Collections)</em></td>
<td>Number of collations in the view (hints at view complexity)</td>
</tr>
<tr>
<td>Ascending/Descending <em>(NIF Collections)</em></td>
<td>Indicates how the collation is sorted, and by which field</td>
</tr>
<tr>
<td>View HDB/Full <em>(NIF Collection Users)</em></td>
<td>Database handle for user &amp; full server access for the database where the view in contained – good for correlating data</td>
</tr>
<tr>
<td>Open By <em>(NIF Collection Users)</em></td>
<td>Indicates the process name, PID &amp; TID of thread that has view opened</td>
</tr>
</tbody>
</table>
Section IV – Private Memory

Introduction

Private Memory can also be important in troubleshooting issues, particularly when high memory usage occurs in process-private memory. Important sections in Private Memory include:

<table>
<thead>
<tr>
<th>Lesson</th>
<th>See Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lesson I - TLS Mapping</td>
<td>28</td>
</tr>
<tr>
<td>Lesson II - Open Documents</td>
<td>29</td>
</tr>
<tr>
<td>Lesson III - Process Heap Memory</td>
<td>30</td>
</tr>
<tr>
<td>Lesson IV - Top 10 Memory Block Usage</td>
<td>32</td>
</tr>
</tbody>
</table>

Finding the beginning of Private Memory

- Search on KEYWORD “memcheck” until you come to the beginning of the Memcheck section
- Search on KEYWORD “attach” – this will bring you to the beginning of the Private Memory section where Memcheck attaches to the first process
- Subsequent searches on KEYWORD “attach” will take you to the next process
- Search on KEYWORD “detach” to get to the end of a specific process’ information

How much private memory?

As stated before, the majority of memory usage for a Domino Server should be from shared memory. In contrast, on average, each server task should use considerably less private memory through the Domino Memory Manager, between 50 MB and 100 MB; there are a few exceptions to this rule, including the server task, http task, and router task, which may use upwards of 200-300 MB.

While it may not indicate a server defect, if private memory usage for a task exceeds these thresholds, an investigation should be made to determine the nature of this memory usage, and if any configuration changes are required. Under these cases, you should call into Support to assist in this investigation.
Lesson I – TLS Mapping

How to Find This Section

• KEYWORD “TLS Mapping” – find the appropriate Process ID and Thread ID

----- TLS Mapping ------

NativeTID | VirtualTID | PrimalTID
[ nSERVER:0514:0510] | [ nSERVER:0514:0002] | [ nSERVER:0514:0002]
[ nSERVER:0514:0504] | [ nSERVER:0514:0004] | [ nSERVER:0514:0004]
[ nSERVER:0514:05d4] | [ nSERVER:0514:0005] | [ nSERVER:0514:0005]
[ nSERVER:0514:0600] | [ nSERVER:0514:0006] | [ nSERVER:0514:0006]
[ nSERVER:0514:0604] | [ nSERVER:0514:0007] | [ nSERVER:0514:0007]
[ nSERVER:0514:0608] | [ nSERVER:0514:0008] | [ nSERVER:0514:0008]
[ nSERVER:0514:060c] | [ nSERVER:0514:0009] | [ nSERVER:0514:0009]
[ nSERVER:0514:0610] | [ nSERVER:0514:000a] | [ nSERVER:0514:000a]
[ nSERVER:0514:06ac] | [ nSERVER:0514:000b] | [ nSERVER:0514:000b]
[ nSERVER:0514:06b0] | [ nSERVER:0514:000c] | [ nSERVER:0514:000c]
[ nSERVER:0514:06b4] | [ nSERVER:0514:000d] | [ nSERVER:0514:000d]
[ nSERVER:0514:06b8] | [ nSERVER:0514:000e] | [ nSERVER:0514:000e]
[ nSERVER:0514:06c0] | [ nSERVER:0514:000f] | [ nSERVER:0514:000f]
[ nSERVER:0514:06f4] | [ nSERVER:0514:0010] | [ nSERVER:0514:0010]
[ nSERVER:0514:06f8] | [ nSERVER:0514:0011] | [ nSERVER:0514:0011]
[ nSERVER:0514:06fc] | [ nSERVER:0514:0012] | [ nSERVER:0514:0012]
[ nSERVER:0514:0700] | [ nSERVER:0514:0013] | [ nSERVER:0514:0013]
[ nSERVER:0514:0704] | [ nSERVER:0514:0014] | [ nSERVER:0514:0014]
[ nSERVER:0514:0708] | [ nSERVER:0514:0015] | [ nSERVER:0514:0015]
[ nSERVER:0514:070c] | [ nSERVER:0514:0016] | [ nSERVER:0514:0016]
[ nSERVER:0514:0710] | [ nSERVER:0514:0017] | [ nSERVER:0514:0017]

How to Use This Section

The **TLS Mapping** is crucial for mapping virtual thread ID used throughout Memcheck with the correct physical thread ID used in other portions of NSD such as call stacks.

You should consult the **TLS Mapping** when you wish to determine which virtual thread ID a particular physical thread is running under (i.e. you have **physical Thread ID**, but not **virtual Thread ID**). You can also use this section if you know the virtual thread ID and need the physical thread ID, but this can also be found in the **Resource Usage Summary** along with all the open resources for that virtual or physical thread.

What are Virtual Threads?

Virtual thread IDs are numbers generated by Domino that allow for an additional abstraction layer between Domino and OS threads. This is done primarily to facilitate scalability for the server when dealing with network I/O. For many server tasks, a given physical thread will constantly be switching from one virtual thread ID to another during normal operations.
Lesson II – Open Documents

How to Find This Section
• KEYWORD “Open Documents” – find the appropriate process

How to Use This Section
Formatted in the same manner as the Open Document Sections from shared memory, this section lists open documents that are opened in private memory. You should use this section when determining how many notes a specific process has open. While the Resource Usage Summary also lists all these documents, that section does not give an indication as to the scope each note. You can use this section to establish the scope of each note if needed.
Lesson III – Process Heap Memory

How to Find This Section
• KEYWORD “Process Heap Memory” (ND6.5.4 and earlier) – find the appropriate process
• KEYWORD “Process Heap Memory Stats” (ND6.5.5 and later) – find the appropriate process

Original Format (ND6.5.4 and earlier)

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE</td>
<td>Count</td>
</tr>
<tr>
<td>SIZE</td>
<td>ALLOC</td>
</tr>
<tr>
<td>FREE</td>
<td>FRAG</td>
</tr>
<tr>
<td>OVERHEAD</td>
<td>%used</td>
</tr>
<tr>
<td>%free</td>
<td></td>
</tr>
</tbody>
</table>

Process Heap Memory:

<table>
<thead>
<tr>
<th>Pool Type</th>
<th>TYPE</th>
<th>Count</th>
<th>SIZE</th>
<th>ALLOC</th>
<th>FREE</th>
<th>FRAG</th>
<th>OVERHEAD</th>
<th>%used</th>
<th>%free</th>
</tr>
</thead>
<tbody>
<tr>
<td>S-DPOOL</td>
<td>719</td>
<td>376963072</td>
<td>211225436</td>
<td>165542364</td>
<td>0</td>
<td>239594</td>
<td>56%</td>
<td>43%</td>
<td></td>
</tr>
<tr>
<td>VPOOL</td>
<td>1</td>
<td>64512</td>
<td>456</td>
<td>62428</td>
<td>0</td>
<td>1636</td>
<td>0%</td>
<td>96%</td>
<td></td>
</tr>
<tr>
<td>POOL</td>
<td>187</td>
<td>2740396</td>
<td>1786784</td>
<td>889772</td>
<td>0</td>
<td>70088</td>
<td>65%</td>
<td>32%</td>
<td></td>
</tr>
<tr>
<td>Overall</td>
<td>719</td>
<td>376963072</td>
<td>210273236</td>
<td>166494564</td>
<td>0</td>
<td>311318</td>
<td>55%</td>
<td>44%</td>
<td></td>
</tr>
</tbody>
</table>

Newer Format (ND6.5.5 and later)

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE</td>
<td>Count</td>
</tr>
<tr>
<td>SIZE</td>
<td>ALLOC</td>
</tr>
<tr>
<td>FREE</td>
<td>FRAG</td>
</tr>
<tr>
<td>OVERHEAD</td>
<td>%used</td>
</tr>
<tr>
<td>%free</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pool Type</th>
<th>TYPE</th>
<th>Count</th>
<th>SIZE</th>
<th>ALLOC</th>
<th>FREE</th>
<th>FRAG</th>
<th>OVERHEAD</th>
<th>%used</th>
<th>%free</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static-DPOOL</td>
<td>12</td>
<td>6291456</td>
<td>3795788</td>
<td>2489080</td>
<td>0</td>
<td>9486</td>
<td>60%</td>
<td>39%</td>
<td></td>
</tr>
<tr>
<td>VPOOL</td>
<td>2</td>
<td>130808</td>
<td>8994</td>
<td>117628</td>
<td>0</td>
<td>4210</td>
<td>6%</td>
<td>89%</td>
<td></td>
</tr>
<tr>
<td>POOL</td>
<td>3</td>
<td>86348</td>
<td>58790</td>
<td>24468</td>
<td>0</td>
<td>3114</td>
<td>68%</td>
<td>28%</td>
<td></td>
</tr>
<tr>
<td>Overall</td>
<td>12</td>
<td>6291456</td>
<td>3653692</td>
<td>2631176</td>
<td>0</td>
<td>16810</td>
<td>58%</td>
<td>41%</td>
<td></td>
</tr>
</tbody>
</table>

Field Label | Description
-------------|--------------------------------------------------
TYPE         | Indicates the various types of pools that are allocated. For private memory, the overall total is currently calculated incorrectly, so you should examine the stats for “S-DPOOL”, or “Static DPOOL” (other pool types such as POOL and VPOOL are actually sub-allocated from S-DPOOL, so this stat will reflect overall usage).

SIZE         | Indicates the total amount of private memory allocated by the Domino Memory Manager, not all of which may be in use. This total should be no higher than about 100 MB (see note on next page).

ALLOC        | Indicates the amount of the “SIZE” that is actually in use (or sub-allocated) at any given time.

%used        | Indicates the percentage used of memory (ALLOC/SIZE). It’s a good thing to have percentages in the 90% range, not a bad thing. The higher the %used, the better – we want to be making good use of what we have allocated.

Continued on next page
**Lesson III – Process Heap Memory, Continued**

**How to Use This Section**

Similar to the “Shared Memory” Summary, this section can be used to establish the total amount of private memory Domino has allocated for a process, which is augmented by the Top 10 [Process] Block Usage sections.

The information provided in this section of Memcheck is identical in nature to the Shared Memory pool section, only this time memory is private to each process. Use this section of Memcheck when dealing with high private memory usage, or some problem resulting from high private memory usage or high private handle counts. For instance:

- if a crash is due to high private memory usage
- if you suspect a leak in private memory
- if you want to evaluate the amount and efficiency of private memory usage

As mentioned earlier, each server task should use considerably less private memory usage (50-100 MB) than shared memory (1.0-1.2 GB); there are a few exceptions to this rule, including the server task, http task, and router task, which may use upwards of 200-300 MB. While it may not indicate a server defect, when private memory exceeds these thresholds, the nature of this memory usage should be investigated.
Lesson IV – Top 10 Process Memory Blocks

How to Find This Section
• KEYWORD “Top 10” – look for appropriate process in [brackets]

Original Format (ND6.5.4 and earlier)

```
#------
Top 10 [ nSERVER:0514] Memory Block Usage:
<table>
<thead>
<tr>
<th>Type</th>
<th>TotalSize</th>
<th>Handles</th>
<th>Type</th>
<th>Handles</th>
<th>TotalSize</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x4129</td>
<td>2883584</td>
<td>11</td>
<td>0x0130</td>
<td>95</td>
<td>34580</td>
</tr>
<tr>
<td>0x29c5</td>
<td>349084</td>
<td>1</td>
<td>0x0380</td>
<td>67</td>
<td>31222</td>
</tr>
<tr>
<td>0x29c4</td>
<td>131072</td>
<td>1</td>
<td>0x030a</td>
<td>47</td>
<td>15604</td>
</tr>
<tr>
<td>0x29c2</td>
<td>65406</td>
<td>1</td>
<td>0x0f02</td>
<td>22</td>
<td>2420</td>
</tr>
<tr>
<td>0x29c3</td>
<td>131072</td>
<td>1</td>
<td>0x0275</td>
<td>22</td>
<td>2554</td>
</tr>
<tr>
<td>0x29c1</td>
<td>65406</td>
<td>1</td>
<td>0x0910</td>
<td>20</td>
<td>5332</td>
</tr>
<tr>
<td>0x0130</td>
<td>34580</td>
<td>95</td>
<td>0x4129</td>
<td>11</td>
<td>2883584</td>
</tr>
<tr>
<td>0x0149</td>
<td>32604</td>
<td>6</td>
<td>0x0149</td>
<td>6</td>
<td>32604</td>
</tr>
<tr>
<td>0x0380</td>
<td>31222</td>
<td>67</td>
<td>0x0f5d</td>
<td>2</td>
<td>220</td>
</tr>
<tr>
<td>0x024b</td>
<td>24908</td>
<td>19</td>
<td>0x0128</td>
<td>1</td>
<td>10480</td>
</tr>
</tbody>
</table>
```

Newer Format (ND6.5.5 and later)

```
<@@ ------ Notes Memory Analyzer (memcheck)...->
Top 10 [ nSERVER: 09d8] Memory Block Usage... ------ @@>

<table>
<thead>
<tr>
<th>Type</th>
<th>TotalSize</th>
<th>Handles</th>
<th>Typename</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x4129</td>
<td>20447232</td>
<td>39</td>
<td>BLK_LOCAL</td>
</tr>
<tr>
<td>0x0a04</td>
<td>10595772</td>
<td>162</td>
<td>BLK_NKT</td>
</tr>
<tr>
<td>0x028b</td>
<td>3327954</td>
<td>53</td>
<td>BLK_FOLDERREPLPLOPS</td>
</tr>
<tr>
<td>0x0910</td>
<td>1999180</td>
<td>1126</td>
<td>BLK_SRV_NAMES_LIST</td>
</tr>
<tr>
<td>0x093c</td>
<td>1219526</td>
<td>242</td>
<td>BLK_SRV_HASH_TBL</td>
</tr>
<tr>
<td>0x024b</td>
<td>1131334</td>
<td>19</td>
<td>BLK_OPENED_NOTE</td>
</tr>
<tr>
<td>0x0221</td>
<td>930818</td>
<td>96</td>
<td>BLK_NEW_NOTE</td>
</tr>
<tr>
<td>0x0130</td>
<td>562418</td>
<td>1545</td>
<td>BLK_TLA</td>
</tr>
<tr>
<td>0x0149</td>
<td>548834</td>
<td>101</td>
<td>BLK_PHTCHUNK</td>
</tr>
<tr>
<td>0x030a</td>
<td>319190</td>
<td>1</td>
<td>BLK_LOOKUP_THREAD</td>
</tr>
</tbody>
</table>
```

Field Label Description

<table>
<thead>
<tr>
<th>Field Label</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>The hexadecimal block type designation used by the component that allocated the block</td>
</tr>
<tr>
<td>TotalSize</td>
<td>Amount of private memory allocated across all blocks of a given type</td>
</tr>
<tr>
<td>Handles</td>
<td>Number of blocks allocated for a given type (each block uses one handle)</td>
</tr>
</tbody>
</table>

Continued on next page
Lesson IV – Top 10 Process Memory Blocks, Continued

How to Use This Section

Use this section for a quick method of establishing process-private block usage broken down by block type. This ONLY shows block usage, not total pool usage. Hence, if a pool is not well utilized, you will not see the problem here. Under those cases, you should consult the Process Heap Memory Summary and collect memory dumps (opening a ticket with Support). You should come here:

- if you suspect that a private block is leaking
- if you get "Out of Private Handles" errors
- if you want to see private block usage

Just as in shared memory, look for either a large amount of total memory usage for one block type, or a large count of one block type. Again, large amounts of private memory over 100 MB should be a concern, with a few exceptions such as server, http, or router (which may use upwards of 200-300 MB).
Section V – Resource Usage Summary

Introduction  No doubt you have noticed that this documentation has repeatedly mentioned the importance of the Resource Usage Summary. As has already been eluded, this section of Memcheck lists all the major resources broken out per thread ID (virtual and physical thread). The format of this section is cleanly organized, allowing for one-stop shopping. This is perhaps the most important section of Memcheck next to memory usage. With the exception of memory usage, you will spend most of your time here. Important output is:

• VThread (Virtual Thread) To PThread (Physical Thread) Mapping
Resource Usage Summary – VThread Info

--- Resource Usage Summary ---

** Process [ nserver:289c ]

.. SOBJ: addr=0x0130004, h=0xf0104002 t=8128 (BLK_PCB)
.. SOBJ: addr=0x022d2628, h=0xf010400f t=8a18 (BLK_NET_PROCESS)
.. SOBJ: addr=0x00000001, h=0x00000000 t=8310 (BLK_NIF_PROCESS)
.. SOBJ: addr=0x02670338, h=0xf01c0036 t=0a22 (BLK_LOCINFO)
.. SOBJ: addr=0x00000000, h=0xf01c0036 t=0901 (BLK_SERVER_PROCESS)
.. SOBJ: addr=0x02670338, h=0xf01c0036 t=0901 (BLK_SERVER_PROCESS)
.. SOBJ: addr=0x010ede50, h=0xf01c0043 t=1381 (BLK_SCT_MGR_CONTEXT)
.. SOBJ: addr=0x00000001, h=0x00000000 t=8310 (BLK_NIF_PROCESS)
.. SOBJ: addr=0x022d2a84, h=0xf0104018 t=8912 (BLK_SERVER_PROCESS)
.. SOBJ: addr=0x022d2894, h=0xf0104028 t=880f (BLK_CLIENT_PROCESS)
.. SOBJ: addr=0x0010f0c3c, h=0xf01c000f t=030f (BLK_NIF)
.. SOBJ: addr=0x02670338, h=0xf010400f t=8a18 (BLK_NET_PROCESS)
.. SOBJ: addr=0x00000000, h=0xf01c0036 t=0901 (BLK_SERVER_PROCESS)
.. Task: TaskID=[ 70:  8392], PRThread: [ nserver:289c:   77]
.. TaskVar: id: [ 70:  8392], tranID=5457, Func=55, st=6, by: CN=Rob Gearhart/O=SET
.. Database: D:\Lotus\DominoR65\Data\mayhem.nsf

** VThread [ nserver:289c:   77]

.. Mapped To: PThread [ nserver:289c:10684]
.. Description: Server for Rob Gearhart/SET on TCPIP

--- Continued on next page ---
Resource Usage Summary – VThread Info, Continued

How to Use This Section

Use this section when you want to know what databases, views, notes or files a physical thread has open. You can find:

- **Database** – establish the database name
- **DBH** – correlate the instance of the database user against other sections of Memcheck
- **By** – to determine the user of the database
- **view** – establish any open views
- **doc** – establish a list of notes open by the thread (keep an eye on class)
- **file** – establish a list of any files the thread had open through Domino file management

Keep in mind that this list of resources is specific to the virtual thread ID, which a suspect physical thread happens to be mapped to at that moment. Not all of these resources were originally opened or even in use by this physical thread at the time of the crash. However, this should not affect your investigation.

Often times, there will be multiple databases, views, etc opened by the thread, only one of which may responsible for the problem. This means that you will need to make an educated guess at to which resource was actually involved in a problem (based in part on arguments on the stack).

Avoid This Pitfall

Be careful not to jump to conclusions regarding a problem database, view or document. Just because a database, view or document was open at the time of the crash doesn’t mean the root-cause of the crash is due to accessing that resource. Not every database involved in a crash is corrupt, in fact most aren’t. There are many cases where a crash or hang is a result of conditions that are completely unrelated to the resource being used. Always approach a crash or hang with a grain of salt. Get an idea from the call stack about the nature of the crash.
Section VI – Correlating Memcheck Output

Introduction

We have seen throughout Memcheck Anatomy, as well as in this unit, how Memcheck lists the same information in multiple contexts. Knowing how to correlate all this information together to find what you need is very important. The Resource Usage Summary correlates much of the important pieces together, but it is helpful to be familiar with how to correlate this directly. This section will discuss how to go about tying certain pieces of information together.

<table>
<thead>
<tr>
<th>Lesson</th>
<th>See Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lesson I – Finding Physical/Virtual Thread</td>
<td>38</td>
</tr>
<tr>
<td>Lesson II – Correlating Database Handle with Open Documents</td>
<td>39</td>
</tr>
<tr>
<td>Lesson III – Correlating View Information</td>
<td>40</td>
</tr>
</tbody>
</table>
Lesson I – Finding Physical/Virtual Thread

PIDs/TIDs  When using a text editor, be sure to use process ID's and thread ID's for locating the process you are troubleshooting, since there can be multiple processes with the same name, like AMGR, Replica, etc. When using a thread ID, you will almost always be starting with a physical (native) thread ID and working your way back to the virtual thread ID using the TLS Mapping section for the process in question. For instance:

Fatal Call Stack:

```plaintext
### FATAL THREAD 16/20 { nHTTP:113c:4284 }
### FF=0x090df664, PC=0x6027f491, SP=0x090df6e8, stacksize=2432
### EAX=0x00000000, EBX=0x00000000, ECX=0x0017f08c, EDX=0xffffffff
### ESI=0x00000000, EDI=0x00000000, ES=0x00000000, DS=0x00000000
### ESI=0x00000002, EDI=0x00000002, FS=0x00000000, GS=0x00000000
### Flags=0x00010202
### Exception code: c0000005 (ACCESS_VIOLATION)

[1] 0x6027f491 nnotes._Panic@4+625 (90df35c,6000118b,0,0)
[2] 0x6000304b nnotes._LockHandle@12+59 (ffffffff,90df6a4,90df69c,0)
[3] 0x60002f11 nnotes._OSLockObject@4+23 (ffffffff,a,3f,1760000)
[4] 0x082d24dc mayhem.MHMNotesPanic+641 (90df7fc,90df8fc,90df9fc,a)
[5] 0x082d1cd6 mayhem.MHMServerCrash+387 (6e,2,82f22f8,64)
[6] 0x082d1171 mayhem.MHMDispatchMayhem+228 (82c70a0,64,6e,2)
[7] 0x082c13f1 mmmdsapi.MHMDispatchFilter+273 (17603e8,90df08,64,6e)
[8] 0x082c13ee mmmdsapi.MHMProcessRawRequest+317 (17603e8,90df08,1,90df30)
[9] 0x082c10a6 mmmdsapi.HttpFilterProc+31 (17603e8,1,90df08,90df04)
[10] 0x10012a2b nhttpstack.HTDsapiRequest::RawRequest+379 (a4,0,17603a0,10017341)
[11] 0x100174a4 nhttpstack.HTRequestExtContainer::RawRequest+164 (1760a68,17601f3c,6f530,1002544f)
[12] 0x10020d08 nhttpstack.HTSession::StartRequest+1299 (6f530,6000118b,0,0)
[13] 0x10034c13 nhttpstack.HTSession::StartRequest+1299 (6f530,6000118b,0,0)
[14] 0x10043929a nhttpstack.HTWorkerThread::CheckForWork+538 (37dccc4,0,0043e63,2)
[15] 0x1004367e nhttpstack.HTWorkerThread::ThreadMain+126 (37dccc4,0,00436b5b,37dccc4)
[16] 0x1003f03e nhttpstack.HTThreadBeginProc@4+63 (37dccc4,80431472,37dccc4,1003f000)
[17] 0x601c0b5b nnotes._ThreadWrapper@4+251 (0,6f530,6000118b,0,0)
[18] 0x7c57438b KERNEL32.TlsSetValue+240
```

---- TLS Mapping ----

<table>
<thead>
<tr>
<th>NativeTID</th>
<th>VirtualTID</th>
<th>PrimalTID</th>
</tr>
</thead>
<tbody>
<tr>
<td>nHTTP:113c:1852</td>
<td>nHTTP:113c:5</td>
<td>nHTTP:113c:5</td>
</tr>
<tr>
<td>nHTTP:113c:2560</td>
<td>nHTTP:113c:6</td>
<td>nHTTP:113c:6</td>
</tr>
<tr>
<td>nHTTP:113c:2556</td>
<td>nHTTP:113c:7</td>
<td>nHTTP:113c:7</td>
</tr>
<tr>
<td>nHTTP:113c:4152</td>
<td>nHTTP:113c:8</td>
<td>nHTTP:113c:8</td>
</tr>
<tr>
<td>nHTTP:113c:1800</td>
<td>nHTTP:113c:9</td>
<td>nHTTP:113c:9</td>
</tr>
<tr>
<td>nHTTP:113c:2580</td>
<td>nHTTP:113c:10</td>
<td>nHTTP:113c:10</td>
</tr>
<tr>
<td>nHTTP:113c:4284</td>
<td>nHTTP:113c:11</td>
<td>nHTTP:113c:11</td>
</tr>
<tr>
<td>nHTTP:113c:2580</td>
<td>nHTTP:113c:12</td>
<td>nHTTP:113c:12</td>
</tr>
<tr>
<td>nHTTP:113c:2556</td>
<td>nHTTP:113c:13</td>
<td>nHTTP:113c:13</td>
</tr>
</tbody>
</table>
Lesson II – Database Handle

Database Handle Locations

Each user’s DBH is listed in several different sections within Memcheck. In the original format for NSD, you can correlate an open note to its database handle using the following two sections (the info of interest is *squared in blue*):

- Open Databases (Database Handle & Database Name)
- Open Documents (Database Handle & NoteID)

```
D:\Lotus\DominoR65\Data\mayhem.nsf
```

The Newer Format for NSD lists the database name directly in the Open Documents section, as well as the name of the user that has the document opened (*squared in blue*):

```
<[@ ------ Notes Memory Analyzer (memcheck) -> Open Documents (BLK_OPENED_NOTE): ......... @>@>
```

```
DBH  NOTEID  HANDLE  CLASS  FLAGS  FirstItem  LastItem
100  2310  0xa132  0x0001  0x0100  [ 6706: 816]  [ 6706:3120]
103  2306  0xa133  0x0001  0x0100  [ 6707: 816]  [ 6707:2988]
100  2306  0xa134  0x0001  0x0100  [ 6708: 816]  [ 6708:3160]
103  2302  0xa136  0x0001  0x0100  [ 6710: 816]  [ 6710:2988]
103  2302  0xa137  0x0001  0x0100  [ 6711: 816]  [ 6711:3160]
```

```
Open By: CN=John Smith/O=ACME/C=US
Flags2 = 0x0404
Flags3 = 0x0000
OrigHDB = 531
First Item = [ 9471: 836]
Last Item = [ 9471: 1228]
Non-pool size : 0
Member Pool handle=0x24ff, size=2984
```
Lesson III – View Information

View Name and Handle Info

Determining view information can be a bit challenging, since the view name is listed in one place, and the DBH to the database is listed in another, and the name of the database is listed in yet another. The sections that allows you to correlate view name and database name are as follows (the info of interest is squared in blue):

- NIF Collections (ViewNoteID or Collection VB)
- NIF Collection Users (ViewNoteID or Collection VB, Database Handle, Thread ID)
- Open Databases (Database Handle, Database Name, Thread ID)

<@@ ------ Notes Memory Analyzer (memcheck) -> NIF Collections (Time 12:48:35) ------ @@>

```
CollectionVB ViewNoteID UNID OBJID RefCnt Flags Options Corrupt Deleted Temp NS Entries ViewTitle
[  0020e005] 1518 1356a8 358710 1 0x0000 00000008 NO NO NO NO NO NO NO MyNotes
```

<@@ ------ Notes Memory Analyzer (memcheck) -> NIF Collection Users (hash) (Time 12:48:33) ------ @@>

```
CollUserVB ... CollectionVB Remote OFlags ViewNoteID Data HDB/Full View HDB/Full ... Open By
[  00239805]  ... [  0023d005] NO 0x0082 786 1219/1874 1219/1874 ... [ nserver:  09d8:  04ca]
CurrentCollation = 0
[  0013a805]  ... [  00136005] NO 0x00c2 1112 886/785 886/785 ... [ nserver:  09d8:  0266]
CurrentCollation = 0
[  0028d805]  ... [  0020e005] NO 0x00c2 1518 551/1432 551/1432 ... [ nserver:  09d8:  03b0]
CurrentCollation = 0
```

<@@ ------ Notes Memory Analyzer (memcheck) -> Open Databases (Time 12:47:18) ------ @@>

```
D:\Lotus\Domino\Data\HR\projnav.nsf
```

SizeLimit = 0, WarningThreshold = 0
ReplicaID = 862568fe:0019c2ad
bContQueue = NSFPool [120: 7236]
FDGHandle = 0xf01c0928, RefCnt = 3, Dirty = N
DB Sem = (FRWSEM:0x0244) state=0, waiters=0, refcnt=0, nlrdrs=0 Writer=[
CurrentCollation = 0
[ By: [ nsERVER:09d8: 03b0] DBH= 551 ]
```

Rob Gearhart/Elliott Harden

40

HND202 - Lotusphere 2007
Section VII – NSD Checklist

NSD Checklist

Below is a suggested checklist that you as an administrator might put together for looking over an NSD. Needless to say, we cannot capture the entirety of NSD analysis here, but hopefully this checklist can give you a broad overview for what to review:

Call Stacks – locate crashing process/thread
- KEYWORD “fatal” or “panic”
- What is the physical thread ID?
- What was the crash point?

Memcheck – Shared Memory – determine Domino shared memory usage
- KEYWORD “Shared Memory”
- KEYWORD “Top 10 Shared Block Usage”
- KEYWORD “Open Databases”
- KEYWORD “Open Documents”
- KEYWORD “NIF Collections” & “NIF Collection Users”
- KEYWORD “Shared OS” or “MM/OS” to determine Panic Message

Memcheck – Private Memory – determine Domino private memory usage
- KEYWORD “Process Heap Memory”
- KEYWORD “Top 10” Process Block Usage
- KEYWORD “TLS Mapping”

Resource Usage Summary – determine databases, views, documents opened by thread
- KEYWORD “Mapped to: PThread”

Parting Words of Wisdom

What should you as an administrator remember about NSD? It’s one of our most important Domino diagnostics, and tells you information about what a server or client was doing at the time of a problem, and what resources it was doing it do.

While NSD is not the proverbial magic bullet, NSD will get you moving solidly in the right direction. We have given you just a taste of what you can do with NSD. Used in conjunction with other diagnostic files such as CONSOLE.LOG, SEMDEBUG.TXT and relevant OS diagnostics, NSD can give you surprising insight into the nature of a crash or hang, if not the root-cause itself. Once you know how to use NSD to your advantage, you can use it to determine the next steps in troubleshooting a problem, and reduce time to resolution while improving your grasp of the issue.
Section VIII – Case Studies

Case Studies

The following are a series of scenarios intended to improve your working knowledge of NSD. You are given a brief explanation of the scenario along with the NSD from the server.

In each scenario, you are given an NSD file, and in some cases, supporting files, and asked a series of questions intended to guide you through the files. We are not asking you to resolve a problem top to bottom. Instead, we may only ask you what the next step would be in a particular situation. This echoes real life, where you may not be able to definitively solve a problem from one NSD, but where you can use what you see in the file to narrow down the problem, and determine next steps.

See the answer key at the end - but no peeking! The provided NSD files should remain in the lab, and should not be taken with you when you leave the lab.

In addition, we have a special surprise for you; we have provided an alpha version of NSDAnalyzer (part of a tool called ISEW) for you to use with this lab, a tool used to view and analyze an NSD. This is an internal beta provided as a special courtesy for Lotusphere only. No – we won’t give you a sneak copy to take home with you, so don’t bother asking. ☺ However, we are giving you a preview of a tool that we eventually hope to make available to customers for use in their environment. We do not have an ETA for when this may be available.

Laptop sign in: Lotus
Password: password

ISEW sign-in: Lotusphere@us.ibm.com
Password: password

Continued on next page
Section VIII – Case Studies, Continued

Scenario 1

File: scenario_1_nsd.log

Problem: AMGR has crashed on an agent.

Question 1: What is the process ID and thread ID of the crashing thread?

Question 2: What is the note ID and database name for the crashing agent?

Question 3: What is the virtual thread ID for the crashing thread?

Question 4: How would you establish the name of the agent that crashes?

Question 5: Using the call stack for the fatal thread, can you find any documented TN’s in Knowledge base, for instance, an SPR?

Continued on next page
Scenario 2  

**Files:** scenario_2_nsd.log & scenario_2_memory.dmp  

**Problem:** The Domino Server crashed.  

**Question 1:** What is the process ID and thread ID of the crashing thread?  

**Question 2:** Why did the server crash?  

**Question 3:** Using the Shared Memory Pools section of NSD, what is overall Shared Memory usage as allocated by Domino? What is %used? Is this normal?  

**Question 4:** Using the Process Heap Memory section of NSD, what is the overall private memory usage for the process in question? What is the %used? Is the normal?  

**Question 5:** Open the memory dump file, and examine “LotusScript” memory usage for the process in question. What is the overall private memory usage in bytes for LotusScript Usage? Is this normal?  

**Question 6:** Adding the amount of private memory usage from question 4 to that of question 5, how many bytes of private memory is being used by the process in question (through both the Domino MM and LotusScript MM)? Is this normal?  

Continued on next page
Section VIII – Case Studies, Continued

Scenario 3

File: scenario_3_nsd.log
Problem: The Domino Server crashed.

Question 1: What is the process ID and thread ID of the crashing thread?

Question 2: What is unusual about the call stack?

Question 3: Given what little clues you have from the call stack, as well as from the Resource Usage Summary for this thread, what do you think the thread might have been doing at the time of the crash?

(hint1 – what is the one DLL name you can decipher in the call stack?)
(hint2 – under what cases would the http process be executing java?)
(hint3 – what is the user ID of the documents opened by this thread?)

Continued on next page
Scenario 4  

Files: scenario_4_nsd.log, scenario_4_semdebug.txt, scenario_4_console.log  

Problem: HTTP is non-responsive.

Question 1: Examining the semdebug.txt, what semaphore is the point of contention? (Hint: look at the semdebug.txt file as a whole)

Question 2: At what time does the semaphore contention begin?

Question 3: Based on the ownership of this semaphore, does it look like the server is hung?

Question 4: What is the timeframe during which the call stacks in the NSD are dumped? (Hint: look for "DBG(…)" to determine the timestamps in NSD).

Question 5: During the window of time in question 4, which thread owns the semaphore in question? As far as you can tell, how long did this thread have the semaphore locked? What was this thread doing during that time?

Question 6: Using the Memcheck portion of NSD, what database and view was experiencing the Collection Semaphore contention?

Question 7: At the time of the NSD, how many readers and waiters were there for this view? How many waiters are waiting for a read versus a write? What are the PID and TID of the first writer waiting for the semaphore?

Question 8: If you had to guess, what would you say is the problem? What would be your next step?

Continued on next page
Section VII – Case Studies, Continued

Scenario 5
(BONUS)

File: scenario_5_nsd.log

Problem: The Domino Server crashed.

Question 1: What is the crashing process and thread ID?

Question 2: What was this server doing when it crashed?

Question 3: Based on the answer to question 2, where would you go next?

Question 4: Who is the largest user of shared memory, and how much are they using? Is this normal?

Question 5: Why is there so much memory allocated for BLK_OPENED_NOTE? (Hint: Look in the Open Documents section)

Question 6: Which user has these problematic documents opened?

Question 7: What do you think is going on here? Does this appear to be a server defect?

Question 8: Can you recommend a course of action to alleviate this problem?
Student Answer Key – Scenario 1

Scenario 1 Answers

File: scenario_1_nsd.log

Problem: AMGR has crashed on an agent.

Question 1: What is the process ID and thread ID of the crashing thread?
Answer: Process= AMgr, Process ID=0B04, Thread ID= 1288

Question 2: What is the note ID and database name for the crashing agent?
Answer: Database='SomeDB.nsf', Agent NoteID=966

Question 3: What is the virtual thread ID for the crashing thread?
Answer: Virtual Thread ID=0002

Question 4: How would you establish the name of the agent that crashes?
Answer: Use the Admin Client and search the database using the NoteID to pull up the agent design note; from there, you can read the “$TITLE” field to establish the agent name. When searching the database, you will need to convert the NoteID from decimal format (as taken from the NSD) to hexadecimal format (as inputted into the Admin Client). In this case, within the Admin client, you would search the database for NoteID 3C6 (that’s 966 in hex).

Question 5: From the fatal call stack, can you match this against any known SPRs?
Answer: Yes, TN 1187989, SPR HRON65SL9C, using “FormatFromVariant” as our search criteria.

Scenario 1 Details

The note listed with HDB=35 (database handle), class=0200 (agent note) and opened by AGTSIGNER/AGT/ACME (signer of the agent) is the agent note we are interested in. The note with HDB=16, class=0200, and opened by user name SERVER01/SVR/ACME is instance of the same agent note opened by the server in preparation to run the agent (either one gives you the correct note ID).

Continued on next page
Scenario 1: Correlating the Data

Fatal Call Stack:

```plaintext
# Fatal Thread 1/4 [ nAMgr:0b04:1288 ]
# FF=0x0012e644, FC=0x00683e2d, CS=0x00000000, stksize=64
# EAX=0x0000001000, ECX=0x0000000000, EDX=0x3b9c0280
# ES=0x05802050, EDI=0x0000000000, CS=0x0000000000, SS=0x0000000000
# Exception code: c0000005 (ACCESS_VIOLATION)

@ [1] 0x60683e2d nnotes.LsConv::FormatFromVariant +541 (c26268,0,12e4c0,12e4b8)
@ [2] 0x6265a763 nlsxbe.ANotes::ANFormatFromVariant +67 (5802050,12e4c0,12e4b8,12e4bc)
@ [3] 0x6265f0da nlsxbe.ANItem::ANICreateFromVariant+42 (15d3bbcc,5802050,0,15d3bbbc)
@ [4] 0x6264290a nlsxbe.ANItem::ANItem+105 (15d3bbcc,15d385b0,5802050,0)
@ [5] 0x6266e2fc nlsxbe.ANSetProp+252 (12e9b8,15d3874e,12e9d4,57923c4)
@ [6] 0x6264290a nlsxbe.ANSetProp+252 (12e9b8,15d3874e,12e9d4,57923c4)
@ [7] 0x62643309 nlsxbe._ANCLASSCONTROL@16+25 (c26268,108,12e9b8,12e9b8)
@ [8] 0x60059212 nnotes.LSsInstance::AdtCallBack+226 (c26268,626427e0,108,57923a8)
@ [9] 0x60091715 nnotes.LScObjCli::SetPropertyData+117 (5792368,f36,5802050,0)
@ [10] 0x6096d745 nnotes.LScObjCli::SetPropEXP+53 (5792368,36,5802050,0)
@ [11] 0x6000222f nnotes.LSsThread::NRun+668 (58014b0,0,14ad978,0)
@ [12] 0x600231b6 nnotes.LSsThread::Run+182 (58014b0,12eb24,6095176d,c26268)

From Resource Usage Summary:

** VThread [ nAMgr:0b04:1288 ]
.VThread ID: nAMgr:0b04:1288
.Maped To: PThread [ nAMgr:0b04:1288 ]
... SOBJ: addr=0x01c4f850, h=0x010000013 t=c176 (BLK_SDKT)
... SOBJ: addr=0x01c4f6e4, h=0x010000035 t=c275 (BLK_NSFT)
... SOBJ: addr=0x01c4f004, h=0x010000037 t=c30a (BLK_LOOKUP_THREAD)
... SOBJ: addr=0x01c4f27b, h=0x010000041 t=c130 (BLK_TLA)
... SOBJ: addr=0x01c4f884, h=0x010000051 t=c436 (BLK_LSITLS)
... Database: X:\Lotus\Domino\Data\Apps\SomeDB.nsf

```
Scenario 2 Answers

Files: scenario_2_nsd.log & scenario_2_memory.dmp

Problem: The Domino Server crashed.

Question 1: What is the process ID and thread ID of the crashing thread?
Answer: Process = nSERVER, Process ID = 0908, Thread ID = 2692

Question 2: Why did the server crash?
Answer: PANIC: Insufficient Memory (Low Memory Condition)

Question 3: Using the Shared Memory Pools section of NSD, what is overall Shared Memory usage as allocated by Domino? What is %used? Is this normal?
Answer: Domino shared memory usage is 988 MB (1,036,481,020 bytes), at 97% used. Yes, these numbers are both normal; shared memory is below the 1 GB threshold, and memory is well utilized at 97% (this is good). Our problem is not in shared memory.

Question 4: Using the Process Heap Memory section of NSD, what is the overall private memory usage for the process in question? What is the %used? Is the normal?
Answer: Domino Private memory usage is 167 MB (175,636,480 bytes), and is 17% used. While the memory usage is not yet over the 200-300MB threshold (exception case for nserver), the pool utilization is VERY low, indicating a potential problem in private memory usage.

Question 5: Open the memory dump file, and examine “LotusScript” memory usage for the process in question. What is the overall private memory usage in bytes for LotusScript Usage? Is this normal?
Answer: LotusScript total (private) memory usage is 184 MB (193,052,121 bytes). Taken by itself, this usage is still under the threshold of 200 MB. However, LotusScript memory usage usually averages much lower (in the 30 MB range), so this memory usage is abnormally high.

Question 6: Adding the amount of private memory usage from question 4 to that of question 5, how many bytes of private memory is being used by the process in question (through both the Domino MM and LotusScript MM)? Is this normal?
Answer: With both the 167 MB of Domino private memory usage, and the 184 MB of LotusScript private memory usage, the Server process has now exceeded 351 MB of private memory usage, which is greater than the 200-300 MB range, and should be investigated.

Continued on next page
The nSERVER process crashed on insufficient memory availability. Based on the NSD (and memory dump), shared memory usage looks normal. Private memory usage however, in the form of Domino private memory and LotusScript private memory is too high. In addition, NSD indicates that the Domino private memory that has been allocated is not being well used (i.e. it’s going to waste).

Other evidence also exists within the NSD (not discussed in this documentation) that third-party components have allocated an additional 150 MB of private under the nSERVER process. Along with an increased number of server threads (120 threads) resulting in a larger memory footprint, the total amount of private memory usage exceeds 620 MB, which has clearly caused a problem. As we can see, no one component has caused the condition, but the memory usage from the combination of factors has caused a problem.

The next step would be to call such an issue into Support in order to employ additional diagnostics to determine what steps can be taken to address memory usage by each separate component (including examining server configuration and database design).

For historical insight, for this issue, additional sets of data that were collected (including memory dumps). This additional data showed that once the server load decreased after hours, LotusScript memory usage dropped back down to 10 MB. This indicates that the design of agents caused an increased usage of LotusScript memory, rather than a server defect (such as a leak). Other components, such as relation database connectors, were also found to be responsible for high memory usage.

Continued on next page
Scenario 2: Correlating the Data

Fatal Call Stack:

```
### FATAL THREAD 67/135 | nSERVER:0908: 2692
### FP=0x0a5ec70, PC=0x6018eb23, stksize=2424
### EAX=0x010d088c, EBX=0x00000000, ECX=0x00ba0000, EDX=0x00ba0000
### ESI=0x0ae5e904, EDI=0x00001d34, CS=0x0000001b, SS=0x00000023
### DS=0x000000023, ES=0x00000023, FS=0x0000003b, GS=0x00000000 Flags=0x00010202
Exception code: c0000005 (ACCESS_VIOLATION)
```

```
@
[1] 0x6018eb23 nnotes._Panic@0+483 (60980013,0,ae5ebe8,6010ee14)
[2] 0x6018e8ec nnotes._ThreadWait(28 (107,0,0,0)
[3] 0x6010ee14 nnotes._MemoryInit180+212 (0,0,5a4cc160,ae5e5fc)
[4] 0x600e1a4 nnotes.OSInitExts8+52 (0,0,ae5f7c8,62641de2)
[5] 0x600e05e nnotes._OSInit+53 (0,0,9754ff4,0a4cc160)
[6] 0x62641de2 nlsxbe._LsxMsgProc@12+130 (2,5a4c8288,5a4cc160,7e5ff4)
[7] 0x6014fe93 nnotes.DLLNode::Register+35 (5a4cc160,7e5f8a0,6014f59d)
[8] 0x6014fa9 nnotes.LSIClassRegModule::AddLibrary+5 (ae5f8a0,ae5f884,7e5f9a,6014f59d)
[9] 0x6014ff3 nnotes.LSISession::RegisterClassLibrary+19 (6014f59d,ae5f8a0,7e5f9a,ae5f94)
[10] 0x6014f59d nnotes.LSISession::RegisterClassLibrary+141 (e967674,ae5f8a0,7e5f9a,321,9e641f9)
[11] 0x60935b0 nnotes.LSCreateScriptSession@20+125 (e9641f8,6010a0d0,0a2aae4,0)
[12] 0x60935ca nnotes.LSClassScriptInit@4+26 (e9641f8,0,ae5f84,9e640f4)
[13] 0x60935e4 nnotes.LSIDocument::Init+30 (e9641f4,9e641f4,605b08f0,1d7f8ff0)
[14] 0x605b0379 nnotes.AgentRun@16+313 (2ea34ec,e9641f4,0,10)
[15] 0x6014d4c nserver1._ServerRunServerAgent@38+450 (8bd00005,7a340008,49a40870,5d9d7dce)
[16] 0x6010990c nserver1._DbServerApp@22+228 (201510c8,8bd00005,0,237ff8)
[17] 0x60100f1f nserver1._WorkThreadTask@15+10 (01030196,0,7f5f9b10)
[18] 0x60100f1c nserver1._Scheduler@7+763 (0,7f5f9b10,0,10ec39c)
[19] 0x60114924 nnotes._ThreadWrapper@8+212 (0,10cc39c,7f5f9b10,0)
[20] 0x7c57438b KERNEL32.TlsWrapper+240
```

```
Shared OS Fields

Start Time = 27/07/2004 16:19:26
Crash Time = 02/08/2004 16:24:56
Error Message = PANIC: Insufficient memory

Shared Memory:

<table>
<thead>
<tr>
<th>TYPE</th>
<th>Size</th>
<th>Alloc</th>
<th>Free</th>
<th>FRAG</th>
<th>OVERHEAD</th>
<th>Used</th>
<th>%used</th>
<th>%free</th>
</tr>
</thead>
<tbody>
<tr>
<td>S-DPOOL</td>
<td>1036481020</td>
<td>1006951768</td>
<td>287000064</td>
<td>0</td>
<td>1468382</td>
<td>97%</td>
<td>2%</td>
<td></td>
</tr>
<tr>
<td>Overall</td>
<td>1036481020</td>
<td>1006951768</td>
<td>287000064</td>
<td>0</td>
<td>1468382</td>
<td>97%</td>
<td>2%</td>
<td></td>
</tr>
</tbody>
</table>

Continued on next page
Student Answer Key – Scenario 2, Continued

Elevated private memory usage (167 MB)

From scenario_2_memory.dmp (under the “nSERVER” section of the memory dump)

LotusScript section within memory dump for crashing process

Total LotusScript private memory usage for nSERVER=184 MB
Student Answer Key – Scenario 3

Scenario 3 Answers

File: scenario_3_nsd.log
Problem: The Domino Server crashed.

Question 1: What is the process ID and thread ID of the crashing thread?
Answer: Process=nHTTP, Process ID=0c18, Thread ID=0bf0

Question 2: What is unusual about the call stack?
Answer: The call stack looks very short (i.e. it is truncated). This call stack provides very little information about what this thread was doing.

Question 3: Given what little clues you have from the call stack, as well as from the Resource Usage Summary for this thread, what do you think the thread might have been doing at the time of the crash?
Answer: The HTTP server crashed while running a java agent

(hint1 – what is the one DLL name you can decipher in the call stack?)
Answer: jvm.dll
(hint2 – under what cases would the http process be executing java?)
Answer: while running a java agent or servlet
(hint3 – what is the user ID of the documents opened by this thread)
Answer: AGTSIGNER/AGT/ACME

Scenario 3 Details

Based on the one DLL name in the stack trace summary (“jvm”), it appears that this thread was executing some type of java code at the time of the crash. Since the crashing thread is in HTTP, chances are strong that this is either a java agent or a servlet.

By examining the Resource Usage for this thread, we can further narrow down the choices by noting that the user associated with open databases for this thread is AGTSIGNER/AGT/ACME. A java servlet would open any databases with the server’s ID rather than a user ID (ruling out a servlet). We can be nearly certain that this HTTP thread was executing an agent at the time of the crash (this can be confirmed with the use of other diagnostics such as htthr.log files).

Continued on next page
What next? *If this thread is executing an agent, why don’t we see an open document with a “class=0200” under the thread ID in Resource Usage?*

Excellent question - for reasons too detailed to discuss here, any time any server task thread (such as HTTP) runs a java agent, the agent note is opened on one thread, but the actual agent execution occurs on a newly spawned JVM thread. As a result, one of the other threads under HTTP Resource Usage Summary will have an agent note open.

Normally, one would need to employ additional INI settings to correlate which HTTP worker thread spawned the java current agent, along with the name of the java agent. However, if we are lucky, within this NSD, there are a limited number of threads that have an agent note open, so we can build a list of possible candidates.

As it turns out, this is exactly the case! Only two other HTTP threads have any agent notes open, and it just so happens that both of those thread have the *same agent note* open (*NOTEID=462, database=Inspired.nsf*). Hence, we have luckily removed all the guess work in determining which agent is involved in the crash.

Even though we are lacking a clean call stack and have not yet isolated root-cause, we have nonetheless been able to establish a surprising amount of information from this one NSD, once we know what to look for. *The next steps* are to examine the design of the agent, and work on extracting a clean call stack for the crash. Clearly, these next steps require the assistance of IBM Support.

*Continued on next page*
Scenario 3: Correlating the Data

From the NSD - Fatal Call Stack:

```
############################################################
### FATAL THREAD 65/67 [ nHTTP:0c18:0bf0 ]
### FP=0x5b75fc8c, PC=0x746e656d, SP=0x5b75fc54, stksize=56
### EAX=0x2d5e1340, EBX=0x00000001, ECX=0x2d5e1340, EDX=0x6276b2e0
### ESI=0x5b75fd08, EDI=0x1abb3010, CS=0x00000001b, SS=0x000000023
### DS=0x000000023, ES=0x000000023, FS=0x0000000038, GS=0x0000000000 Flags=0x00010202
### Exception code: c0000005 (ACCESS_VIOLATION)
############################################################
```

From the NSD - Resource Usage Summary:

- VThread [ nHTTP:0c18:0fbf ]
  Mapped To: PThread [ nHTTP:0c18:0bf0 ]
  SOBJ: addr=0x28d801f8, h=0xf01054aa t=d1ba (PKG_LSBE+10)
  SOBJ: addr=0x517b16d0, h=0xf01053c0 t=c130 (BLK_TLA)
  SOBJ: addr=0x2c25167c, h=0xf0105342 t=c30a (BLK_LOOKUP_THREAD)
  Database: X:\Lotus\Domino\Data\Dept\Online.nsf
  DBH: 573, By: CN=AGTSIGNER/OU=AGT/O=ACME
- Database: X:\Lotus\Domino\Data\Dept\Support.nsf
- Database: X:\Lotus\Domino\Data\Dept\Standard.nsf
  DBH: 494, By: CN=AGTSIGNER/OU=AGT/O=ACME
- Database: X:\Lotus\Domino\Data\Dept\SSN.nsf
  DBH: 494, By: CN=AGTSIGNER/OU=AGT/O=ACME
- Database: X:\Lotus\Domino\Data\Inspired.nsf
  DBH: 1056, By: CN=AGTSIGNER/OU=AGT/O=ACME

All databases opened by this thread use AGTSIGNER/AGT/ACME ID

DLL indicates thread is likely executing Java of some kind

---

Continued on next page
Student Answer Key – Scenario 3, Continued

From Resource Usage Summary:

** VThread [ nHTTP:0c18:0fbe]
.Mapped To: PThread [ nHTTP:0c18:1144]
.. SOBJ: addr=0x29e5120c, h=0xf0105042 t=d1ba {PKG_LSBE+10}
.. SOBJ: addr=0x0f2e03a0, h=0xf0104850 t=c130 {BLK_TLA}
.. Database: F:\Lotus\Domino\Data\Inspired.nsf
...... DBH: 1074, By: CN=AGTSIGNER/OU=AGT/O=ACME
...... doc: HDB=1074, ID=462, U=101, class=0200, flags=0000

** VThread [ nHTTP:0c18:000d]
.Mapped To: PThread [ nHTTP:0c18:0e58]
.. SOBJ: addr=0x0f312278, h=0xf01042ad t=c275 {BLK_NSFT}
.. SOBJ: addr=0x104409f4, h=0xf01043f8 t=c30a {BLK_LOOKUP_THREAD}
.. SOBJ: addr=0x015de788, h=0xf010417c t=c130 {BLK_TLA}
.. SOBJ: addr=0x0f3903bc, h=0xf01044a4 t=c436 {BLK_LSITLS}
.. Database: F:\Lotus\Domino\Data\Inspired.nsf
...... DBH: 597, By: Anonymous
...... doc: HDB=597, ID=462, H=3861, class=0200, flags=0001
...... doc: HDB=597, ID=0, H=2390, class=0001, flags=0000
...... DBH: 1082, By: CN=AGTSIGNER/OU=AGT/O=ACME

Only two HTTP threads that have an agent note open, both are the same agent.
Student Answer Key – Scenario 4

**Files:** scenario_4_nsd.log, scenario_4_semdebug.txt, scenario_4_console.log

**Problem:** HTTP is non-responsive.

**Question 1:** Examining the semdebug.txt, what semaphore is the point of contention? (Hint: look at the semdebug.txt file as a whole)

*Answer:* The FRWSEM 0x030B Collection semaphore

**Question 2:** At what time does the semaphore contention begin?

*Answer:* 08/02/2005 11:27:22:09 AM EDT. The previous semaphore timeouts occur 15 minutes prior, and do not implicate the Collection Semaphore.

**Question 3:** Based on the ownership of this semaphore, does it look like the server is hung?

*Answer:* No. The timeouts are not continuous (there are breaks of a few seconds to a few minutes between timeouts), and the owner of the Collection Semaphore changes at least every 2 minutes. If the server were hung, the timeouts would be continuous over long periods of time (say 30-60 minutes) with no change in ownership of the semaphore.

**Question 4:** What is the timeframe during which the call stacks in the NSD are dumped? (Hint: look for “DBG(…)” to determine the timestamps in NSD).

*Answer:* The call stacks were dumped between 11:33:21 AM and 11:33:48 AM. This provides a 27-second window that can be compared to semdebug.txt

**Question 5:** During the window of time in question 4, which thread owns the semaphore in question? As far as you can tell, how long did this thread have the semaphore locked? What was this thread doing during that time?

*Answer:* PID=0F9C, TID=0FE4. This thread had the collection semaphore locked from 11:33:10:49 AM EDT to 11:34:04:15 AM EDT, or about 54 seconds. The thread was performing a DBLookup on the view, which means it locked the semaphore for a read.

**Question 6:** Using the Memcheck portion of NSD, what database and view was experiencing the Collection Semaphore contention?

*Answer:* Database=profiledb.nsf, view=(UserID)|wUserID

Continued on next page
**Student Answer Key – Scenario 4, Continued**

**Scenario 4 Answers (continued)**

**Question 7:** At the time of the NSD, how many readers and waiters were there for this view? How many waiters are waiting for a read versus a write? What are the PID and TID of the first writer waiting for the semaphore?

*Answer:* 27 readers, 13 waiters. Of the waiters, 2 are writers, 11 are readers. The first thread waiting for a write is nHTTP:0f9c:108c.

**Question 8:** If you had to guess, what would you say is the problem? What would be your next step?

*Answer:* There are frequent updates to a view that is experiencing a heavy throughput of readers. The next step is to look at the database design (see side note below).

**Scenario 4 Details**

Based on the pattern of the waiters in the NSD, it appears that writer activity is sprinkled in between heavy reader activity (20+ readers). Based on the call stacks for the various view readers, there are a large number of DBLookups being performed on the view in question. There is probably a design issue in the application with too many DBLookups being performed on a large view intermixed with frequent updates. In order to confirm a consistent pattern to the reader and writer activity, you will need to collect multiple sets of data (NSD and Semdebug) over different slow downs.

The next step is to examine the design of the view to determine how long it may take to update, and to investigate how pervasive the lookups are to this view. As a sanity check, you could investigate if it appears that all activity on the Web Server is slow, or only pages that interact with this view (either directly or indirectly). You will probably find the latter.

In this particular case, it turns out the customer overloaded the view with too many lookups in addition to the fact that they were frequently adding documents to this view. The recommendation was to reduce the number of lookups in the application, split out the lookups across multiple views to reduce contention, and batch the addition of documents on a more scheduled basis, say once every hour or so instead of once every two or three minutes. These types of performance issues could and should be identified with stress testing *prior* to a rollout into production.

*Side Note:* The review of the application is an investigation that needs to be lead primarily by the creator and maintainer of the application (that’s you), where direction and assistance can be provided by Support. In this case, based on the data “so far”, there is no evidence of a Server defect.

*Continued on next page*
Scenario 4: Correlating the Data

From scenario_4_semdebug.txt:

08/02/2005 11:32:32:33 AM EDT sq="00008AFA" THREAD [0F9C:0010-01E4] WAITING FOR FRWSEM 0x030B Collection semaphore (@02079108) (R=4,W=0,WRITER=0000:0000,1STREADER=0F9C:1008) FOR 30000 ms

Most pervasive semaphore timeout

08/02/2005 11:33:10:49 AM EDT sq="00008BFA" THREAD [0F9C:000D-108C] WAITING FOR FRWSEM 0x030B Collection semaphore (@02079108) (R=30,W=0,WRITER=0000:0000,1STREADER=0F9C:0FE4) FOR 30000 ms

Change in ownership

08/02/2005 11:33:10:52 AM EDT sq="00008BFB" THREAD [0F9C:023A-035C] WAITING FOR FRWSEM 0x030B Collection semaphore (@02079108) (R=30,W=0,WRITER=0000:0000,1STREADER=0F9C:0FE4) FOR 30000 ms

08/02/2005 11:33:24:25 AM EDT sq="00008C06" THREAD [0F9C:0023-0FAC] WAITING FOR FRWSEM 0x030B Collection semaphore (@02079108) (R=28,W=0,WRITER=0000:0000,1STREADER=0F9C:0FE4) FOR 30000 ms

08/02/2005 11:33:40:49 AM EDT sq="00008C2F" THREAD [0F9C:000D-108C] WAITING FOR FRWSEM 0x030B Collection semaphore (@02079108) (R=27,W=0,WRITER=0000:0000,1STREADER=0F9C:0FE4) FOR 30000 ms

Window of time during which NSD dumped call stacks (see next page)

Continued on next page
From scenario_4_nsd.log:

```
<table>
<thead>
<tr>
<th>INFO</th>
<th>PID</th>
<th>PPID</th>
<th>UID</th>
<th>STIME</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBG(134c) 11:33:21</td>
<td>0000 0000</td>
<td>0</td>
<td>???</td>
<td>0</td>
<td>???</td>
</tr>
<tr>
<td></td>
<td>0008 0000</td>
<td>65535</td>
<td>???</td>
<td>[ ]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0148 0008</td>
<td>0</td>
<td>07/27 11:37:19</td>
<td>[ ]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0164 0148</td>
<td>0</td>
<td>07/27 11:37:21</td>
<td>[ ]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0160 0148</td>
<td>0</td>
<td>07/27 11:37:22</td>
<td>[ ]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0198 0160</td>
<td>0</td>
<td>07/27 11:37:22</td>
<td>[ ]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>* 0334 0198</td>
<td>0</td>
<td>07/27 11:37:26</td>
<td>[ ]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>044c 0334</td>
<td>0</td>
<td>07/27 11:37:28</td>
<td>[ ]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>076c 044c</td>
<td>0</td>
<td>07/27 11:37:37</td>
<td>[ ]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-&gt; 116c 044c</td>
<td>0</td>
<td>08/02 10:44:55</td>
<td>[ ]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-&gt; 0750 116c</td>
<td>0</td>
<td>08/02 10:50:34</td>
<td>[ ]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DBG(134c) 11:33:48</td>
<td>memcheck -k cur -p 0x116c -p 0x750 -p 0x9f9c -p 0x14a4 -p 0x370 -p 0xecc -p 0x11bc -p 0x11b8 -p 0x11c8 -p 0xf60 -p 0x358 -p 0x120c -p 0x125c -p 0x1160 -p 0x12a4 -p 0x1134 -p 0x1100 -p 0x1170 -p 0x928 -p 0x118c -p 0x12e0 -p 0xfdc -p 0x858 -d err -o</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Call Stack for 1st Reader of Collection Semaphore from time of NSD:

```plaintext
############################################################
### thread 55/96: [ nHTTP:0f9c:0fe4]
### FP=10e57d58, PC=77f883a3, SP=10e57d3c, stkbase=10d60000, stksize=262144
############################################################
@
[ 1] 0x77f883a3 ntdll.RtlTimeToTimeFields+421 (0,0,600a1dfd,0)
[ 2] 0x7c59a25a KERNEL32.SetLocalTime+790 (0,f9c,15c77ce8,10e57d9c)
[ 3] 0x60001076 nnotes._OSLockSpin+848 (15c77ce8,1600000,10e57e70,15c4761e)
[ 4] 0x60003705 nnotes._UNLockReadFRWSem+85 (15c77ce8,10e57d8,6000679b,0)
[ 5] 0x60005ebb nnotes._BUFSUnLockContainer+12+43 (0,10e57d4,1,1)
[ 6] 0x60006b70 nnotes._NSFSUnLockPinWithHateInfo+16+249 (2098395,60c31b20,0,0)
[ 7] 0x60006583 nnotes._NSFSlotUnPinWithHateInfoCtx+16+227 (2098395,60c31b20,10e57e70,0)
[ 8] 0x60008856 nnotes._NSFSlotUnPinCtx+8+22 (2098395,60c31b20,10e57e70,0)
[ 9] 0x60048c0b nnotes._BTGetLongEntry+429 (209bae8,2098395,60c31b20,0)
[10] 0x60009f73 nnotes._BTGetLocalEntry+330 (22bae8,2098395,60c31b20,0)
[11] 0x6000a9e1 nnotes._BTGetLocalEntry+330 (136b8e,320fb869,60c31b20,9f57)
[12] 0x6000a902 nnotes._BTGetLocalEntry+330 (209bae8,2098395,60c31b20,9f57)
[13] 0x6000e73 nnotes._BTGetLocalEntry+330 (209bae8,60c31b20,0,0)
[14] 0x6000c6a nnotes._CollectionNavigate+238 (0,10e581e4,5,1)
[15] 0x6005d86 nnotes._DBLookup+3+330 (5,10e583d0,6,0)
[16] 0x6005ee33 nnotes._DBLookup+3+330 (2a2,10e583d0,20790e8,0)
[17] 0x60098206 nnotes._NIFFindByKeyExtended+24+590 (2a2,10e583d0,2,10e585dc)
[18] 0x11ea1f7c nDBnotes._DBDLookup+32+2014 (2c3c7bd4,1,0,ffff)
[19] 0x11ea131e nDBnotes._DBDPerformFunction+858 (1546956,4d60cd54,0,10002)
[20] 0x6005bf60 nnotes._DBDFunction+36+2014 (1546956,230d7ce8,0,5)
[21] 0x004f4419 NINOTES.CmdHandler::HandleOpenFormCmd+473 (2c3d8014,0,0,2c3d8014)
[22] 0x0040021d NINOTES.CmdHandler::PrivHandle+269 (2c3d8014,0,0,56a4014)
[23] 0x0043e24b NINOTES.CmdHandler::PrivHandle+123 (2c3d8014,473cd814,2c3d8014,2c3d6014)
[24] 0x0043e36d NINOTES.CmdHandler::Handler+221 (2c3d8014,56a4014,6003e4c0,10e5e300)
[25] 0x004384e4 NINOTES.Cmd::Execute+58 (1cb493fc,1cb49118,0,67049c)
[26] 0x004f7e33 NINOTES._InotesHTTPProcessRequest+1715 (1cb4940c,1cb493fc,1cb49118,0)
[27] 0x004f7eaf NINOTES._InotesHTTPProcessRequest+47 (1cb4940c,3,19980084,6f80)
[28] 0x10001414 nhttpstack.HTNotesRequest::ProcessRequest+36 (1cb49118,1cb48f00,0,0)
[29] 0x1000104e1 nhttpstack.HTRequestExtContainer::ProcessRequest+545 (5,1999de10,1bc69204,0)
[30] 0x10001d3fa nhttpstack.HTRequest::ProcessRequest+1722 (e7f9e96,e7f9bd4,7f9e8e,7070612f)
[31] 0x100021aa nhttpstack.HTSession::StartRequest+858 (e7f9eb0,e7f9bd4,0,1)
[32] 0x10002b29 nhttpstack.HTWorkerThread::CheckForWork+441 (3,e7f9bd4,10028280,100282aa)
[33] 0x10002b97 nhttpstack.HTWorkerThread::ThreadMain+87 (e7f9bd4,0,0,0)
[34] 0x60115bca nnotes._ThreadWrapper+4+212 (0,0,0,0)
[35] 0x7c57b368 KERNEL32.CreateProcessW+310 (0,0,0,0)
```

Continued on next page
From `Memcheck->NIF Collections` section of scenario_4_nsd.log:

<table>
<thead>
<tr>
<th>View NoteID</th>
<th>View Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>2430</td>
<td>a27dc59d</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CollectionVB Collection ViewNoteID</th>
<th>UNID</th>
<th>NS Entries</th>
<th>ViewTitle</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**First Waiter**

"mode=W" indicates first waiter is waiting to lock the semaphore for a write.
(Note the interleaved nature of the waiting readers and writers)

**Waiters**

1. **Mode=W** indicates the first waiter is waiting to lock the semaphore for a write.
2. The mode=W Waiter semaphore information is shown.
3. The current readers are shown along with their semaphore information.
4. The number of current readers is indicated.

**Current Readers**

**Collection Semaphore Information**

- **CollIndex** = [3507: 2291]
- **Collation 0**: BufferSize 15, Items 1, Flags 0
- 0: Ascending, by KEY, "$1", summary# 1
- ResponseIndex [65535: 65534]
- NoteIDIndex [65535: 65534]
- UNIDIndex [65535: 65534]
Scenario 5

Answers

File: scenario_5_nsd.log

Problem: The Domino Server crashed.

Question 1: What is the crashing process and thread ID?
Answer: Process Name=nserver, Process ID= 09d8, Thread ID= 0d30

Question 2: What was this server doing when it crashed?
Answer: Allocating Memory in an attempt to create a network buffer in order to service a client request. The key here is, we crashed trying to allocate memory. We don’t see a PANIC message to this affect, so some investigation is required to see if it really is a low memory condition.

Question 3: Based on the answer to question 2, where would you go next?
Answer: Shared Memory Usage, Top 10, and if needed, Private Memory Usage for nserver.exe.

Question 4: Who is the largest user of shared memory, and how much are they using? Is this normal?
Answer: BLK_OPENED_NOTE is using over 600 MB, more than the UBM. This is not good, and is certainly not normal!

Question 5: Why is there so much memory allocated for BLK_OPENED_NOTE?
(Hint: Look in the Open Documents section)
Answer: While there are not a large number of documents opened (say around 60 notes), several dozen of these notes are VERY large at 24 MB a piece, which is resulting in high memory usage. It only takes a few of these to kill the server. Notice that all of these documents are in various mail files.

Question 6: Which user has these problematic documents opened?
Answer: The BES server (Blackberry) - CN=BES/O=ACME/C=US

Continued on next page
Question 7: What do you think is going on here? Does this appear to be a server defect?

Answer: It looks like a large document was sent out as an e-mail distribution. The BES server has opened these rather large documents to scan them, inflating memory usage. This is not a server defect.

Question 8: Can you recommend a course of action to alleviate this problem?

Answer: Yes. Use SERVER_MAX_NOTEOPEN_MEMORY_MB to restrict memory usage that results from the server opening documents on behalf of the client.

In this case, it appears that a large document with several embedded images was sent out as a mass mailing, where the various embedded images give the document an effective size of 24 MB. The BES server, in its attempt to keep all the end users up to date on their e-mail has opened these rather large documents to scan them, which is ballooning memory usage. If you open up enough of these documents at one time, memory usage will hit critical mass causing a crash.

It’s likely that just given normal user activity, memory usage would not have been an issue, since the transactions of opening the large documents would have been relatively short and randomized. However, the BES server is scanning all these notes at once, and since they are large, they remain open in memory a bit longer that normal (even though they are “fast opens”). The remote BES server compounds the problem.

This behavior is not a server defect per se, since there is not a leak or a mismanagement of opened document. The same effect could be accomplished by a poorly written agent that opens many documents at once. However, this behavior does emphasis a scaling issue with Domino in a 32-bit environment. This scaling issue is not restricted to Domino; many enterprise applications have encountered limitations in the 32-bit address space, especially on platforms such as W32, which only provides a 2 GB user address space as the default.

Continued on next page
The ultimate solution is to move to a 64-bit model, which greatly expands the process address space. However, this move is not trivial; as it stands, the Domino 64-bit version is still under development.

The short term solution is to use the INI parameter designed (as a stop-gap measure) for just this problem. See DCF document #1198511, titled “Can the number of BLK_OPENED_NOTE blocks be limited?” which discusses the use of the INI parameter SERVER_MAX_NOTEOPEN_MEMORY_MB.

In this case, the server crashes when BLK_OPENED_NOTE memory climbs above a few hundred MB, so we can set the value as follows:

\[
\text{SERVER\_MAX\_NOTEOPEN\_MEMORY\_MB}=150
\]

This will limit the number of these large documents (should they occur) to no more than about six at a time. However this may affect performance, since we force the other “open note” transactions to pending until memory usage drops sufficiently.

**Warning:** As stated above, this parameter may affect performance. In addition, this parameter only addresses the case where the Server opens a note on behalf of a remote Native Notes Client (NRPC). This INI will not be effective for notes that are opened by the server for its own purposes (such as agents, etc), and will not affect notes opened via other protocols, such as HTTP, IMAP, SMTP, etc. This is a fun one, eh?

See the following Knowledge Base Articles for more details:

- #1198511 – “Can the number of BLK_OPENED_NOTE blocks be limited?”
Scenario 5 – Correlating the Data

From the NSD - Fatal Call Stack:

Sample excerpt of memory trace:
```
  AllocDBlock@12+14 (60c1e9c4,2018,695fcd8)
  _OSAllocDBlock@12+20 (60c1e9c4,2018,695fcd8)
  _DPoolAlloc@16+606 (ba0014,2018,8a03,5dab28f1)
  _AllocObject@24+75 (ba0014,60c42a40,5dab28e3,2018,695f874,b46cfc)
  _OSMemoryAllocate@12+141 (ba0014,2012,695fe2c)
  _NetAllocBuffer@16+291 (86e40001,0,ffff,695fe2c)
  _StartServer@8+852 (909c0590,57c4006a)
  _DbServer@8+3616 (909c0590,57c4006a)
  _DbListen@8+2302 (7dca0ef0,18780001)
  _Scheduler@4+763 (0)
  _ThreadWrapper@4+212 (0)
  KERNEL32.lstrcmpiW+197
```

Exception code: c0000005 (ACCESS_VIOLATION)

Invalid stack frame detected: Unable to read process memory for frame

From NSD - Shared Memory Stats:

Sample excerpt of shared memory usage:
```
<table>
<thead>
<tr>
<th>TYPE</th>
<th>Count</th>
<th>SIZE</th>
<th>ALLOC</th>
<th>FREE</th>
<th>FRAG</th>
<th>OVERHEAD</th>
<th>%used</th>
<th>%free</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static-DPOOL</td>
<td>434</td>
<td>1694498816</td>
<td>1643729036</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>97%</td>
<td>2%</td>
</tr>
<tr>
<td>Overall</td>
<td>434</td>
<td>1694498816</td>
<td>1643729036</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>97%</td>
<td>2%</td>
</tr>
</tbody>
</table>
```

Invalid stack frame detected: Unable to read process memory for frame

High Shared Memory Usage at ~ 1.6 GB (above the 1 GB avg)

Utilization is good - Top 10 will show where all this memory is going
From NSD - Top 10 Shared Memory Block Usage:

<table>
<thead>
<tr>
<th>Type</th>
<th>TotalSize</th>
<th>Handles</th>
<th>Typename</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x824b</td>
<td>640926118</td>
<td>9824</td>
<td>BLK_OPENED_NOTE</td>
</tr>
<tr>
<td>0x82cd</td>
<td>523522048</td>
<td>133</td>
<td>BLK_UBMBUFFER</td>
</tr>
<tr>
<td>0x8219</td>
<td>127381734</td>
<td>3843</td>
<td>BLK_UNK</td>
</tr>
<tr>
<td>0x841c</td>
<td>81158214</td>
<td>29909</td>
<td>BLK_VARRAY_CHUNK</td>
</tr>
<tr>
<td>0x8252</td>
<td>61865984</td>
<td>59</td>
<td>BLK_NSF_POOL</td>
</tr>
<tr>
<td>0x8a03</td>
<td>21757340</td>
<td>2654</td>
<td>BLK_NETBUFFER</td>
</tr>
<tr>
<td>0x82d2</td>
<td>21645750</td>
<td>5512</td>
<td>BLK_BUCKETDESCPAGE</td>
</tr>
<tr>
<td>0x834a</td>
<td>17301504</td>
<td>17</td>
<td>BLK_GB_CACHE</td>
</tr>
<tr>
<td>0x8946</td>
<td>12582912</td>
<td>12</td>
<td>BLK_SRV_COMPRESS</td>
</tr>
<tr>
<td>0x8234</td>
<td>11288576</td>
<td>2756</td>
<td>BLK_BITMAP</td>
</tr>
</tbody>
</table>

From the Open Documents (BLK_OPENED_NOTE) section:

<table>
<thead>
<tr>
<th>DBH</th>
<th>NOTEID</th>
<th>HANDLE</th>
<th>CLASS</th>
<th>FLAGS</th>
<th>IsProf</th>
<th>#Pools</th>
<th>#Items</th>
<th>Size</th>
<th>Database</th>
</tr>
</thead>
<tbody>
<tr>
<td>363</td>
<td>938262</td>
<td>0x1ca5</td>
<td>0x0001</td>
<td>0x4100</td>
<td>No</td>
<td>389</td>
<td>1970</td>
<td>25442934d</td>
<td>d:\notedata\mail\randrews.nsf</td>
</tr>
</tbody>
</table>

Open By: CN=BES/O=ACME/C=US
Flags2 = 0x0400
Flags3 = 0x0000
OrigHDB = 363
First Item = [ 7333: 848]
Last Item = [ 6763:63948]
Non-pool size : 0
Member Pool handle=0x1ca5, size=65406
Member Pool handle=0x1c12, size=65406
Member Pool handle=0x1c0b2, size=65406
Member Pool handle=0x1c0a2, size=65406
Member Pool handle=0x1c0a6c, size=65406
Member Pool handle=0x1c0c27, size=65406
Member Pool handle=0x1c0be2, size=65406
Member Pool handle=0x1a39, size=65406
Member Pool handle=0x1a69, size=65406
Member Pool handle=0x1b69, size=65406
Member Pool handle=0x1b7b, size=65406
Member Pool handle=0x1b82, size=65406
Member Pool handle=0x1c0fc, size=65406
Member Pool handle=0x1cd4, size=65406
Member Pool handle=0x1b74, size=65406
Member Pool handle=0x1a6b, size=65406
Member Pool handle=0x1ad3, size=65406
Member Pool handle=0x1b81, size=65406
Member Pool handle=0x1ad2, size=65406

Mail File
Large Document (size is 24 MB, 389 pools, 1970 fields)
Document requires many, many pools (too many pools)