OOM Lab Part 1 - SimpleOOM

Locate the OOM folder on the C drive

WordPad or Notepad will be used to open several of the files.

You will also need to open a Command Prompt window (ie, DOS prompt)

In the OOM folder, open the file named
Simplejavacore.20100727....txt

and answer the following questions:

1. What is the exception thrown?
What field name contains the exception?
The exception is "java/lang/OutOfMemoryError", found in the field 1TISIGINFO

2. What is the zOS release in use?
See the field 2XHOSLEVEL
3. What is the java version in use?
See the field 1CIJAVAVERSION
4. What java 'command' was executed?
See the field 1CICMDLINE
5. What is the Application or Class name in use?
(this is included in 1CICMDLINE)The answer is "SimpleOOM"

6. Locate the MEMINFO section: This section of a javacore includes MEMory INFOrmation

7. How much heap is available? See the field 1STHEAPFREE
How much is allocated? See the field 1STHEAPALLOC
What can you determine based on these numbers?
In this case, you can tell that the entire java heap contents have been exhausted
because HEAPFREE is zero. You also can tell the maximum heapsize is defined as xA00000.
The heap will be in HEX, in decimal it is 10485760 bytes (or 10 Meg).
NOTICE that this value shows up in multiple fields :
1CICMDLINE java -Xmx10m SimpleOOM
and
2CIUSERARG -Xmx10m

this helps determine the heap size was passed to the jvm as an argument

8. Find the 'Current Thread Details' and review the java stack:
What is the field of interest? 1XMCURTHDINFO

9. What source code line number was last executed when the exception occurred?
First, look at the field 3XMTHREADINFO, this is the 'Main' thread running, but to find out what
code is executing, look at 4XESTACKTRACE . This is a very simple testcase, containing only a single stack
entry. To determine the line number, let's break down the stack entry:
at SimpleOOM.main(SimpleOOM.java:7(Compiled Code))

The first part is the class and method name (SimpleOOM is the

class, main is the method)
Next is the name of the source file for the class (SimpleOOM.java)
The number following is the source code line number (7)
So, the last line to execute was number 7 . Keep in mind, though, that many times the code executing
when an OutOfMemory occurs may be just a victim, not the actual problem source.

10. Is the method JIT (Just in Time) compiled? What might that mean?
on the stack trace entry we just covered, note the (Compiled Code) at the end of the line.
This indicates the method within this class (method main within class SimpleOOM) has been JIT
compiled. This means the method has been "compiled" into assembler code, which helps improve
performance.

NEXT, Open the file named SimpleGCOut using Notepad or WordPad

11. What is this output? And how is it captured?

I'll help you here: if you've not seen this before,
this is a Garbage Collection trace, captured by passing -verbosegc
when java is initialized. If you returned to the javacore, you could find -verbosegc within
the 1CICMDLINE field, or within a 2CIUSERARG field. In this case, I've thrown you a curve -
you won't find it within the javacore, as this trace was collected on a second invocation of the
class.

The key gc entry type is the 'Global' entry, this is what performs the actual garbage collection

12. How frequently are Global GC's occurring?
Make note of the 'intervalms' field: discount the first entry found ,
<gc type="global" id="1" totalid="1" intervalms="0.000">
Why? Because the FIRST entry will always be ZERO

13. By browsing the 'global' entries , and noting the intervalms field, does GC activity seem excessive?
Keep in mind 'intervalms' represents the interval in milliseconds.
In general, this would be considered somewhat excessive due to the frequency.
How would you correct this? Consider increasing the maximum heap size. But, there
may be more to investigate if more storage doesn't change the results.

Given the simplicity of this example, you may be able to review the GC trace
activity easily. But, what if this were data collected over a long period of
time? We'll answer this in a moment...

BUT, what if -verbosegc wasn't in use? Let's move to the Snap trc

files...

Snap trc files contain raw JVM trace data. It is formatted using the command (using an IBM JVM)

java com.ibm.jvm.format.TraceFormat <Snap....trc>

The default output file will be named with a suffix of '.fmt' appended to the Snap trace input filename

So, let's look at the the file named
SimpleSnap.20100727.141018.33620725.0001.trc.fmt
Use WordPad or Notepad. Scroll to the bottom of the file.
The key entry to look for is J9Allocateobject()

18:10:18.926222374*00000000 j9mm.100 Event J9AllocateObject() returning NULL! 32 bytes requested for object of class 0x225026b0 from memory
space '' id=0x0

14. What size of an object was requested but failed?
In this case, the last allocation that failed was a request for 32 bytes.

Next, we'll try using PMAT (Pattern Modeling and Analysis Tool)
From a DOS window,
cd OOM
and enter the following at the prompt:

```
java -Xmx900m -jar ga439.jar SimpleGCOut
```

Note the GCStats, etc...
Move your cursor to the top of the tool bar, and select the option "Graph View All"

15. Notice the vertical bar to the right? What is that displaying?
(hint:recall the Gc's occured milliseconds apart)

Take some time to investigate the various display options.

****
OOM Lab Part 2 - StartThreads
With what you learned in Lab 1,
Try answering the same questions about the Javacore, GC trace, and Snap trace for Lab 2. All the files for Lab 2 will begin
with 'Start', ie, Startjavacore...., StartSnap.. etc.

When reviewing the javacore, take note of any differences, especially in the 1TISIGINFO field, and
the 1XMCURTHDINFO 'Current Thread Details' field

16. What does Native Method mean?
the top of the stack for the failing thread is:

```
4XESTACKTRACE          at java/lang/Thread.startImpl(Native Method)
```

Native Method indicates the code in question is NOT java, but is C, C++, Assembler, etc.

Review the ENVINFO

17. is there anything there that warrants further investigation?
    One worth considering would be
     -Xss500k
     This defines a stack size of 500K for every thread started. This *may* or *may not* be interesting, depending on the application.

Review the gc tracing in StartGCOut

?????????
   Why are there no entries?
    Remember, Garbage Collection doesn't execute unless it's needed. So, this application doesn't appear to use much heap space.

  So, PMAT won't help..... there is no data available for it to use.

Review StartSnap....trc.fmt

?????????
   There's no object allocation failure, here, either..

HINT: If you didn't notice, return to Startjavacore...txt , and review the exception.. and make note of the
    FULL exception text

Detail "java/lang/OutOfMemoryError" "Failed to create a thread: retVal -1073741830, errno 112" received

This error is actually a NATIVE storage OOM error, requiring a zOS tdump to analyze further.
Capturing a tdump requires use of :

-Xdump:system:events=systhrow,filter=java/lang/OutOfMemoryError