Article
Ning Zhang · Oct 26, 2016 5m read

Excessive memory usage caused by processes invoking JAVA 6 on AIX 7

It has been noticed that some customers running JAVA programs (for example, FOP) on AIX would see the server eventually running low then out of memory. Customer would notice the system pages heavily and user experience becomes bad. And the server would crash when out of memory.

When the problem happens, we can see in ipcs a lot of shared memory segment marked for deletion (Capital D at the beginning of MODE section). This means they will not disappear until the last process attached to the segment detaches it.

The problem can be easily recreated using the following command from UNIX prompt on a server running AIX 7, for example:

/usr/java6_64/jre/bin/java -Xrs -classpath /scratch3/gcooper/ggc170refp750/dev/java/lib/JDK16/cachegateway.jar:/scratch3/gcooper/ggc170refp750/dev/java/lib/JDK16/cachejdbc.jar:/usr/java6_64/jre/bin com.intersys.gateway.JavaGateway 62001 2>&1

It will generate a shared memory segment marked for deletion that can be checked by running ipcs in another UNIX terminal.

A truss on the JAVA process shows the following system calls related to shared memory.

39191692: 100599669: shmget(-1, 536870912, 8576) = 199229473

39191692: 100599669: shmctl(0x000000000BE00021, 0x00000000000000C8, 0x0000010010154940) = 0x0000000000000000

39191692: 100599669: shmat(0x000000000BE00021, 0x0000000000000000, 0x0000000000000000) = 0x0A00000000000000

39191692: 100599669: shmctl(0x000000000BE00021, 0x0000000000000000, 0x0000000000000000) = 0x0000000000000000

It shows the shared memory segment being created, attached to, and then marked for deletion, without being detached.

However, the same command on a server running AIX 6 will not have this problem. A truss shows this at the same place

66388396: 109445345: kmmap(0x0000000000000000, 67108864, 3, 17, -1, 0x0000000000000000, 0x0900000000E3F538) = 0x0700000000000000

So instead of creating a shared memory segment using shmget() and shmctl(), it simply did a mmap() (and in a smaller size).

The root cause was eventually traced down to change SR7 made to JVM version 1.6. The specific change that resulted in this issue is, on AIX systems, the Java heap is now by default allocated with 64K pages, instead of 4K pages default pre-SR7. When the 64K allocation is used and the AIX system is configured to support "Large Page size", the JVM will use shmat() to allocate the heap from shared memory segments. Here ' s a comparison of default memory parameters for JVM before and after SR7 using java –version and java - verbose:sizes

Pre-SR7	Post-SR7
java version "1.6.0-internal" Java(TM) SE Runtime Environment (build pap6460-2007081901) IBM J9 VM (build 2.4, J2RE 1.6.0 IBM J9 2.4 AIX ppc64-64 jvmap6460-20070817 13537 (JIT enabled) J9VM - 20070817013537BHdSMr JIT - dev20070817 1300 GC - 20070815AA) -Xmca32K RAM class segment increment -Xmco128K ROM class segment increment -Xmns0K initial new space size -Xmnx0K maximum new space size -Xms4M initial memory size -Xmos4M initial old space size -Xmox64M maximum old space size	java version "1.6.0" Java(TM) SE Runtime Environment (build pap6460sr10-2011120801(SR10)) IBM J9 VM (build 2.4, JRE 1.6.0 IBM J9 2.4 AIX ppc6 jvmap6460sr10-2011120796808 (JIT enabled, AOT e J9VM - 20111207096808 JIT - r920111107 21307ifx1 GC - 20110519AA) JCL - 2011110402 -Xmca32K RAM class segment increment -Xmco128K ROM class segment increment -Xmns0K initial new space size -Xmnx0K maximum new space size -Xms4M initial memory size -Xmos4M initial old space size

-Xmx64M memory maximum -Xmr16K remembered set size -Xlp4K large page size available large page sizes: 4K 64K 16M 16G -Xmso256K OS thread stack size -Xiss2K java thread stack initial size -Xssi16K java thread stack increment -Xss512K java thread stack maximum size	-Xmox512M maximum old space size -Xmx512M memory maximum -Xmr16K remembered set size -Xlp64K large page size available large page sizes: 4K 64K 16M -Xmso256K operating system thread stack size -Xiss2K java thread stack initial size -Xssi16K java thread stack increment -Xss512K java thread stack maximum size

Notice the different in version as well as the default on –Xlp and –Xmx.

So we did a test with –Xlp4K specified and the shared memory problem went away. A truss found the JAVA process using mmap() system call instead of shmget().

The problem seems to be associated with AIX 7 because AIX 7 comes with a post-SR7 JVM while AIX 6 comes with a pre-SR7 JVM. But it is really about JVM version and any customer who updated their JVM would run into this problem, despite AIX version. That poses a threat to our (at least Trak) customers that they will be hit by this when and if they upgrade JVM on their AIX servers. And we should solve this for them before they got hit.

So a quick work-around to the original problem would be to specify –Xlp4k when invoking JAVA so we do not create shared memory segments. That should solve Trak customers ' problem for now.

But what really needs to be done is that we should talk to whoever develops JVM for AIX and ask them to put a shmdt() into the code so the process would not stop the shared memory segment from disappearing. This shouldn ' t be a problem since it was already marking it for deletion.

One other thought is about –Xmx, memory maximum. It looks like JVM would allocate heap the size of –Xmx. For JVM 1.6 SR7 and above, the default got changed from 64MB to 512MB. We should really check into how much our application actually need and specify it to reduce memory consumption.

#Performance