|  |
| --- |
| **Description** |

|  |  |
| --- | --- |
| |  | | --- | | Name: nt126004 Date: 07/15/2002  FULL PRODUCT VERSION :  java version "1.4.0"  Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.0-b92)  Java HotSpot(TM) Client VM (build 1.4.0-b92, mixed mode)  FULL OPERATING SYSTEM VERSION : Microsoft Windows 2000  [Version 5.00.2195]  A DESCRIPTION OF THE PROBLEM :  A file opened for read-only access and memory mapped using  the map method of FileChannel cannot be deleted even when  the channel is closed.  STEPS TO FOLLOW TO REPRODUCE THE PROBLEM :  1.Compile and run the source code  2. Run it as follows: java testFileDelete filename  where filename is the name of the file to be deleted.  EXPECTED VERSUS ACTUAL BEHAVIOR :  Expected: List the contents of the file specified on the  command line and delete it.  Actual: Lists the contents of the file, however fails to  delete it.  ERROR MESSAGES/STACK TRACES THAT OCCUR :  Could not delete file testdel.txt  REPRODUCIBILITY :  This bug can be reproduced always.  ---------- BEGIN SOURCE ----------  import java.io.\*;  import java.nio.\*;  import java.nio.channels.\*;  import java.nio.charset.\*;  public class testFileDelete {  public static void main(String[] args) {  FileInputStream fis = null;  if (args.length < 1) {  System.err.println("Usage: java testFileDelete <filename>");  System.exit(1);  }  File f = new File(args[0]);  try {  // Open the file  fis = new FileInputStream(f);  } catch (FileNotFoundException ex) {  System.err.println("Error! " + ex.getMessage());  System.exit(2);  }  try {  // Get a channel from the stream  FileChannel fc = fis.getChannel();  // Map the file into memory  MappedByteBuffer bb = fc.map(FileChannel.MapMode.READ\_ONLY, 0, (int)fc.size());  // Do something interesting here. For this example, just print the  // contents of the file.  // Decode the file into a char buffer, so we can print the contents.  Charset cs = Charset.forName("8859\_1");  CharsetDecoder cd = cs.newDecoder();  CharBuffer cb = cd.decode(bb);  // Now print it out to standard output  System.out.print(cb);  // Close the channel and the stream  fc.close();  // Close the input stream even though closing the  // channel should do this  fis.close();  } catch (IOException ex) {  System.err.println("Error! " + ex.getMessage());  System.exit(3);  }  // Done processing file. Now delete it.  boolean deleted = f.delete();  if (!(deleted)) {  System.err.println("Could not delete file " + f.getName());  System.exit(2);  }  }  }  ---------- END SOURCE ----------  CUSTOMER WORKAROUND :  FileChannel.map seems to be causing this problem. A  possible workaround is to not use the NIO package.  (Review ID: 153724)  ======================================================================  Name: nt126004 Date: 07/15/2002  FULL PRODUCT VERSION :  java version "1.4.0"  Java(TM) 2 Runtime Environment, Standard Edition (build  1.4.0-b92)  Java HotSpot(TM) Client VM (build 1.4.0-b92, mixed mode)  FULL OPERATING SYSTEM VERSION :  Microsoft Windows 2000 [Version 5.00.2195]  A DESCRIPTION OF THE PROBLEM :  If a file has been memory mapped (using  java.nio.channels.FileChannel.map), File.deleteOnExit does  not work reliably. It depends on whether the mapped buffer  has been 'finalized' before the JVM exits.  STEPS TO FOLLOW TO REPRODUCE THE PROBLEM :  1. Run the attached code. You have a 1KB 'temporary' file  left behind.  2. If you uncomment the System.gc() line, the subsequent  delete may work (but in my actual code this is unreliable).  EXPECTED VERSUS ACTUAL BEHAVIOR :  I would expect the File.deleteOnExit to always delete the  file. This would require the JVM to unmap any mapped files  before deleteing 'temporary' files. This failure I consider  a bug.  There is a second issue, which is the difficulty of using  File.delete and other methods when we can't control the  timing of unmapping. I will submit a separate RFE for this.  REPRODUCIBILITY :  This bug can be reproduced always.  ---------- BEGIN SOURCE ----------  import java.io.\*;  import java.nio.MappedByteBuffer;  import java.nio.channels.FileChannel;  class TestMemoryMapping  {  public static void main(String[] args)  {  try  {  File f = File.createTempFile("Test", null);  f.deleteOnExit();  RandomAccessFile raf = new RandomAccessFile(f, "rw");  raf.setLength(1024);  FileChannel channel = raf.getChannel();  MappedByteBuffer buffer = channel.map  (FileChannel.MapMode.READ\_WRITE, 0, 1024);  channel.close();  raf.close();  buffer = null;  // System.gc();  if (f.delete())  System.out.println("Temporary file  deleted: "+f);  else  System.out.println("Not yet deleted: "+f);  }  catch (IOException ex)  {  ex.printStackTrace();  }  }  }  ---------- END SOURCE ----------  CUSTOMER WORKAROUND :  Calling System.gc sometimes works.  (Review ID: 153921)  ====================================================================== | |
|  |

|  |
| --- |
| **Comments** |

|  |  |
| --- | --- |
| |  | | --- | | EVALUATION  We cannot fix this. Windows does not allow a mapped file to be deleted. This  problem should be ameliorated somewhat once we fix our garbage collectors to  deallocate direct buffers more promptly (see 4469299), but otherwise there's  nothing we can do about this.  -- ###@###.### 2002/7/16 | |