**.NET Debugging Demos Lab 4: High CPU hang**

**Reproduce the issue**

1. Start the application and browse to All Products, this should take about 5 seconds or more depending on what machine you’re on
2. Watch the CPU usage in task manager while browsing to this page
   * What is the CPU usage like for iisexpress.exe during the processing of this request?

**Watch the .net performance counters**

1. Start monitoring the performance counters for the iisexpress.exe process
2. dotnet-counters monitor -n iisexpress --refresh-interval 1 --counters System.Runtime

**Prepare to gather a memory dump**

1. Prepare to capture a memory dump with procdump or dotnet-dump but don’t hit enter
2. procdump -ma iisexpress.exe

or

dotnet-dump collect -n iisexpress

**Reproduce the issue again and gather the memory dump**

**Note:** Watch the performance counters - specifically the collections - while reproducing

1. Generate load with .\ToolsAndScripts\tinyget.ps1 -url https://localhost:44350/Products -numTimes 5
   * How high does the CPU Usage (%) get?
   * What is the ratio between Gen 0, Gen 1, and Gen 2 GC Count? What should it be in an optimal scenario and why?
   * What can cause a ratio like this?
2. Reproduce the issue again
3. When the CPU usage is high, press enter in the procdump/dotnet-dump window to get the dump - alternatively you can also right-click in task manager to generate a memory dump.

**Open the dump to figure out what it is doing**

1. Open the dump in windbg
2. Load up the symbols and sos

**Verify that you took the dump at the right time**

1. Run !threadpool to see the CPU usage of the system to make sure that you got the dump at the right time
   * What is the CPU Utilization?
   * How does the CPU Utilization shown here correlate to the CPU usage of the process?

**Determine which threads have been consuming most of the CPU time**

1. Run .time to see the uptime and CPU User time of the process
2. Run !runaway to see the User Mode Time for all the threads
   * From !runaway which threads are consuming most of the CPU?
   * How much user mode CPU time was consumed by the process in total? (from .time)

Note: Looking at !runaway output can be a bit tricky for multiple reasons.

* + First off, on a multi processor machine you have to remember that the user mode time (i.e. clock cycles spent in user mode code) is CPU time spent on all processors, therefore the user mode time may add up to more than elapsed time (process uptime).
  + Secondly, the !runaway output shows the total user mode time spent on that thread since it was started. In asp.net for example a thread may be reused for multiple requests so a high user mode time on a thread does not necessarily mean that the requests running on that thread is a big CPU consumer.
  + And finally, some threads like the GC threads (in multi processor, serverGC processes) stay alive for the duration of the process, so they have a higher chance of accumulating a lot of user mode time than worker threads, therefore it is really more interesting to look at two consecutive dumps and compare the differences in user mode time.

1. Pick the threads that have the highest user mode time and look at their stacks
2. ~#s # set thread context, pick a thread number from !runaway
3. kb 2000 # look at the native stack
4. !clrstack # look at the .net stack
   * What are they doing? Can you make an assumption based on this as to what is causing the high CPU usage?
5. [Only on multi-proc machines since they have dedicated GC threads]. Sum up the user mode time for the GC threads in !runaway and divide this by total user mode time from .time.
   * How much of the total user mode time is spent by the GC threads?

**Look at the memory dump to figure out what caused the high CPU in GC**

1. Run ~\* kb 2000 to get all native stacks and search for the thread that triggered the GC (coreclr!SVR::GCHeap::GarbageCollectGeneration)
   * Why was a GC triggered?
   * What type of object was it allocating?
   * What is the thread doing?
   * Could this be causing the bad Gen 0, Gen 1, Gen 2 ratio? Why?
2. Find out what is on the large object heap

**Note:** if you unfortunate you may have caught the GC in the plan or relocate phase in which case the !dumpheap output may not be correct. If that is the case try to make do with what you get from the stacks and !dumpheap -min 85000)

!dumpheap -min 0n85000

* + What is on the large object heap?
  + Is there any pattern in the sizes of the strings?

1. Dump out a few of the strings to see the contents
2. !do <address of string>
   * Based on the stacks, the knowledge of the collection ratios and the string contents, can you determine what went wrong. You might want to do some internet searches on some of the calls on the stacks if you don’t know the answer immediately.

**Verify your assumption in the code and modify the code to resolve the issue**

1. Look at the code for the function you identified
   * What line of code is causing the problem?
   * How do you resolve it? Why does this resolve the problem?
2. Fix the problem and rerun the tests to see the results.