1. We have a class (that runs a thread in its instance) called PeriodicWorker. Are you referring to this, i.e. do you see some (or many) threads named "OpcLabs.PeriodicWorker.Process"? If so, how does it look in "normal" situation, and how does it look when the problem occurs (if it can be determined at that time)?

Instrumentation

Since we do not know that the issue has occurred until after it occurs, we decided to monitor AIT’s requested memory locations continually. Memory is read and values are placed in queue to be logged every 10 milliseconds in the checkIrigRunning() thread, priority set to AboveNormal. All messages logged contain Sync Pulse counter and the DateTime.Now so this log file can be cross-referenced with the Irig log file.

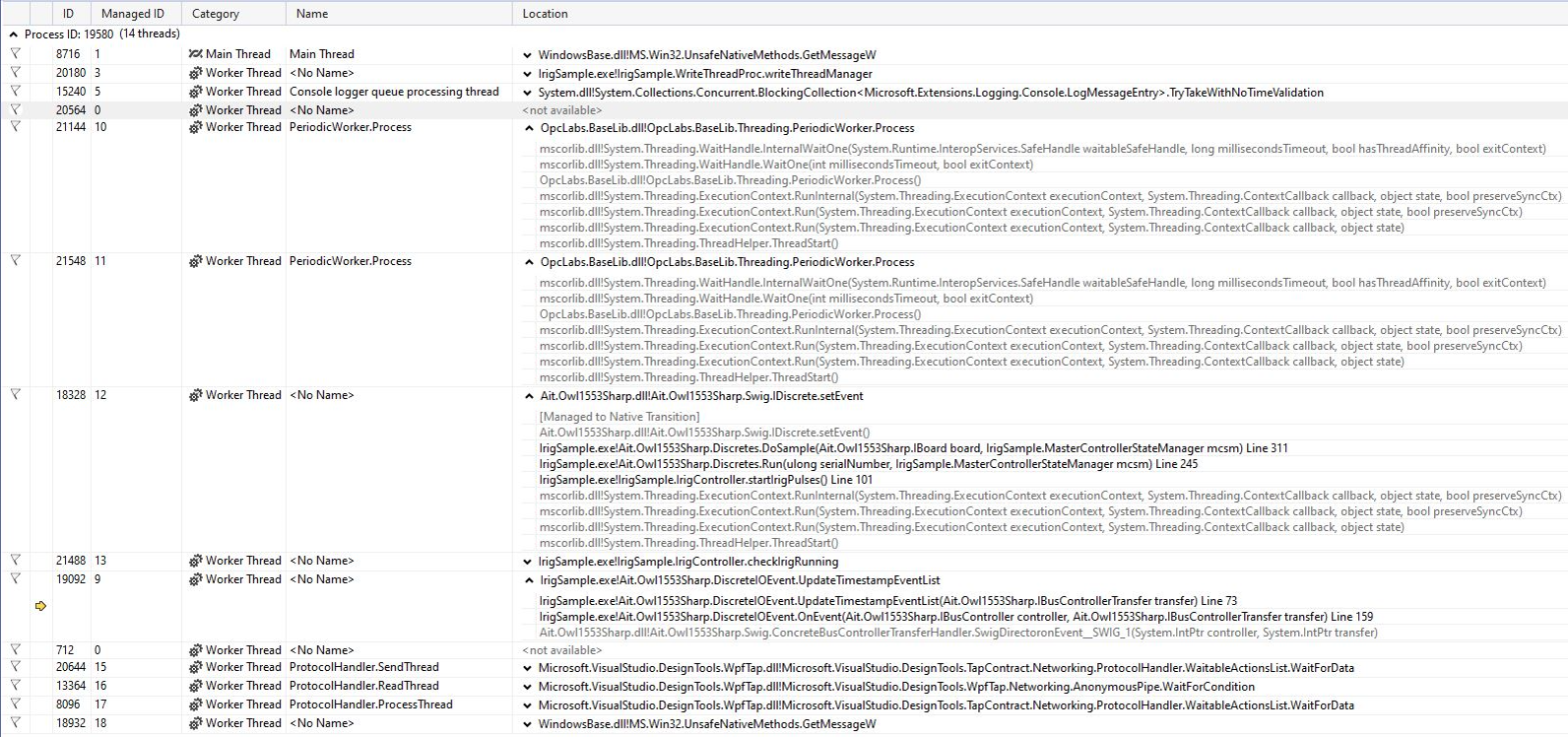
During the OnEvent callback (which should be called every 10ms), each TimeTag event is logged to the Irig log file with the DiscreteIOEventTimes.Count so we can see if the queue is backed up with TimeTag dioEvents to be processed. A value of 1 indicates that the callback is invoked for each interrupt. Higher values indicate that the callback was not invoked and the TimeTags were queued in the Hardware Interrupt service routine.

Results

When the queue becomes backed up with TimeTag events, checkIrigRunning() is also adversely affected. The reading of the memory values appears to take much longer as evidenced by the DateTime.Now logged between “After Sleep” and “After Read”. This is shown by looking at the DiscreteIOEventTimes.Count high values in the Irig log file and comparing the checkIrigRunning() times logged during that same period.

To determine if other threads were similarly affected, I created another simple thread and logged DateTime.Now every 10ms in another file with the Sync Pulse counter. This thread regularly logged approximately every 10ms, except when the queue became backed up. Whatever anomaly is causing the OnEvent method to not be invoked is also having an adverse effect on other threads.

Below shows what was running when the Sync Pulse event queue had more than 12 events in it and the Callback was invoked. Because the Call back was invoked, it is difficult to see what prevented it from being called.



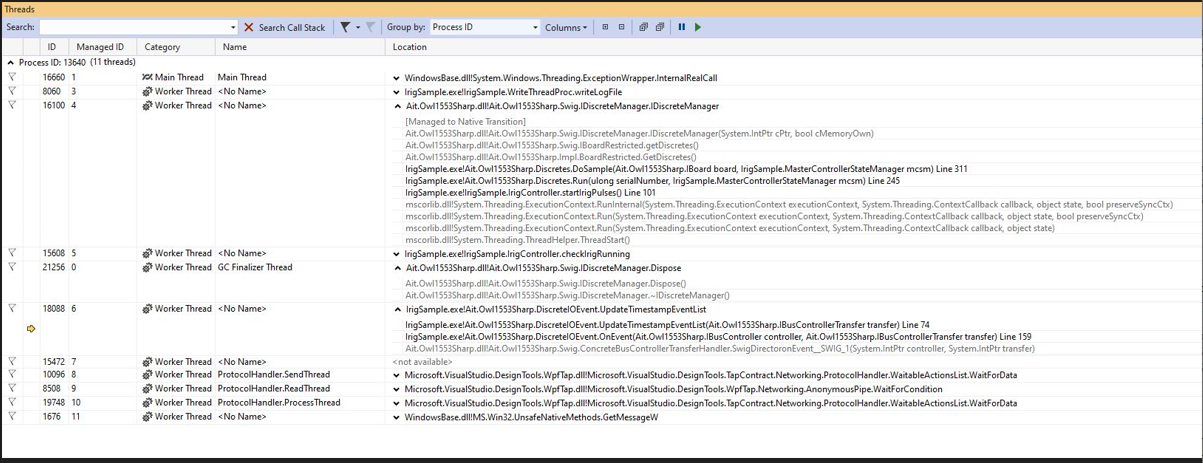
2. Is the general CPU usage on the computer, and CPU usage by the application, "good" (low)?  
CPU usage for the IrigSample application is around 12%. Total CPU usage is 12-13%.

3. The architecture ("block diagram") of the whole system is not fully clear to me. But I assume that whatever dequeues the entries from the "event status queue" in the .NET application must be running on some thread. How is this thread created? Is it with "new Thread()"? Or is a thread from a .NET thread pool, perhaps a Task? Or something else?  
The Callback invoked by AIT’s Hardware Interrupt is an override for AIT.Owl1553Sharp.dll abstract OnEvent discrete event handler. It is enabled when the board is initialized.

4. What is the full .NET Framework version being used by the application on the "good", and "bad" machines? Suggestion is to read and capture System.Runtime.InteropServices.RuntimeInformation.FrameworkDescription from inside the app.

The running Framework are 4.8.4110.0 and 4.8.3928.0 on the good machines.  
 The running Framework is 4.8.9037.0 on a bad machine.  
  
5. Have you tried to break into the debugger when the problem occurs, and look for suspicious symptoms? Mainly, does the number and "structure" of managed threads significantly differ when the problem occurs? (this is repeating a part of question #1, but is more general here)

When EasyUAClient() is not invoked, the events do not back up. Below shows what was running.

   
  
6. Does the issue possibly appear multiple times during a single run of the application, if you let it run long enough- or just once? (If it happens once, it may indicate an issue with .NET JIT compilation)  
The issue appears multiple times during a single run of the IrigSample application, but the times it occurs are random.

7. In the original report we discussed removing the call to client = new EasyUAClient(), the problems stopped occurring. Does this mean that adding "client = new EasyUAClient()", \*without any further operation on that 'client' object\* , is enough to introduce the problem? Or did you actually mean that besides "client = new EasyUAClient()", a whole set of new functionality has also been enabled inside the app, consequently?

Just adding the call to EasyUAClient() appears to cause the problem to occur. We do not subscribe to any tags or write to any tags. Commenting out the call makes the problem disappear.