Problem

Sometimes it takes longer than five seconds to receive an event after it was published. What's going on?

Solution

If the developer-implemented *receive* method of a *Consumer* class takes a large amount of time to complete, this bottlenecks events being sent to that particular consumer. That is, there is a mutex lock somewhere in the TAO notification service making sure the first call to ***push_structured_event*** comp letes before this consumer method is invoked a second time.

The effect this has on other consumer instances is undesireable as well but not horrible most of the time mainly because TAO uses a pool of threads to handle consumer objects instead of a single thread for all consumers. Events being received by other consumers get delayed even if their *receive* metho d is "lightweight".

To get around this the HLA group has stated that non-ACS subsystems should queue the incoming events from their *receive* method and have a separate thread created within their code to process the events later. You should use your best judgement when deciding if your *receive* implementation is "heavy" or not. In general, implementations involving CORBA calls on other objects need to have the event queued. The generic steps needed to process these events later are:

1. Within whatever class you're using (normally a Component implementation but it very well could be something else) define a queue member variable. If the queue class you're using is not thread safe, also define a mutual exclusion lock
2. Define a thread. This thread will loop forever basically just executing the code you now have within your *receive* method or event handler function. The subtle difference is at each iteration of the forever loop it should take an event from the queue and process it
3. Modify your *receive* or event handler function. The code that used to process the event should be placed in the body of the forever loop in the thread from the previous step. receive/handler should now just toss the event on your member queue.
4. Create the thread from the class in **step 1** before invoking Consumer's *consumerReady* function

ACS may help with the decision whether any particular `receive` method should be implemented asynchronously, see NCClassesShouldMonitorReceiv eMethodExecutionTime.

Related articles

- How can more people do development with ACS on the same machine without disturbing each other?
- Which ports are used by ACS?
- Problems connecting to ACS servers on a remote machine: bad /etc/hosts
- Why does the getComponent method of ZLegacy/ACS.ContainerServices return an object of type None?
- Why are some of my print statements not showing up in the container output section of acscommandcenter?