



4/29/2004 11:01 AM



Dr. Samek,

I just finished the FSU class CIS 5930 where we studied hierarchical state machines and quantum programming using your text, PSiCC. As a class we completed a project that was composed of four active objects built for the Windows environment. While we learned a lot in the process we ran into a problem that we were never able to solve. I would appear to us that at least under certain conditions the event pool management is leaking events and eventually the pool will be exhausted. One of our active objects was posting an 80-byte event every 100ms and even when we removed all subscriptions to the event, events were being allocated from the pool but not released. As a class we were unable to track down the problem and now that the class is over, I just wanted to rule out any known problems with the version of QF that was delivered on the CD from the book. I have attached with this email a trace showing the creation, publishing, propagation and annihilation of the event over a short run of the program where there were no subscribers to the event. Please note that the trace behavior is very similar when there are subscribers (at least with respect to exhausting the pool.) Listed in this trace is the myNfree value from the event pool, which clearly shows the pool being depleted. If after reading my description of the problem and looking at the trace listing you could offer any suggestions as to what might cause this problem I would be greatly appreciative.

In closing let me say that I enjoyed your book and your presentation of quantum programming. While I don't do embedded programming these days, I see a lot of potential for using QF for GUI applications. Also let me apologize for not using your forum instead of emailing you directly. It would appear that my employer has blocked the host of your forum.

— **JoAnn Peeler, Principal Design Analyst, Specialty Products Division, Florida, Best Software, Inc.**

continue...





“ Hi JoAnn,

I'm sorry to hear that the class ended with an unsolved problem, which certainly left you wondering if the method is really any good and robust.

I'm not sure if I read your trace correctly. In the absence of subscriptions, `QF::publish()` calls immediately `QF::annihilate()` (see Listing 8.4, line 12). However, even the very first event in your trace is not immediately annihilated after publishing, but rather another event is created. By the same token, `QF::propagate()` should never be called, because the only place `QF::propagate()` is invoked is AFTER dispatching the event. Your trace indicates that `QF::propagate()` is called, which suggests that the event went through an event queue and was dispatched to a state machine.

I guess that your trace might got corrupted because of the multithreading nature of the application. Remember, `printf()` (or `cout`) facility is a SHARED resource. I'd suggest that you check a couple of things:

1. make sure that your application uses the MULTITHREADED runtime library (in VC++ you set it in the "Project Settings" dialog box, "C/C++" Tab, "Code Generation" category, "Use runtime library" option.)
2. make sure that you never overflow/underflow your event after creating it and before publishing it. The risk here is that if you, say, overflow the event, you might modify the NEXT event and thus might destroy the integrity of the event pool.
3. make sure you don't use the event AFTER publishing it.
4. make sure that every event you publish is a subclass of `QEvent`. If it isn't you might overwrite the inherited part that is initialized in `QF::create()`.

One last thing; you can take a look at the Win32 version of the Dining Philosophers, which uses six active objects and runs for days without leaking events. I'd really like to get to the bottom of your problem. Please keep me in the loop.

Regards,
— **Miro Samek**

continue...





4/29/2004 11:01 AM

“ Dr. Samek,

I just wanted to say thank you for replying. Your response made me think about how the event pools are being used and I had forgotten that any one pool can handle multiple event/signal types. Given that, I adjusted my trace to print out the sig value and low and behold, the signals that were causing the leak were not the signals that everyone assumed! Like so many "unsolvable" problems, this problem was easily solved as it turned out that one of the teams had used the Orthogonal Component pattern in their QActive-derived component and they were using Q_NEW to allocate new events to be dispatched to the contained HSM. Unfortunately, they were not releasing the event after the Orthogonal Component had handled the event. Actually, I'm not sure how they would do this without making the QActive derived class a friend of QF since QF::annihilate is a private member. I ended up changing the method to use static memory for the events passed to the orthogonal component and that fixed our leak!

So that everyone understood that QF itself wasn't the culprit, I made sure that every student in the class understood the problem, how it was fixed and each were sent the updated project. Our project has run for a couple of days without exhibiting any more event leakage. The whole thing is very solid!

Thanks again for your work and presentation of HSMs and quantum programming.

— **JoAnn Peeler, Principal Design Analyst, Specialty Products Division, Florida, Best Software, Inc.**

