Subject: Re: "Alternative Multithreading" revisited
Posted by tojocky on Wed, 10 Feb 2010 21:55:39 GMT
View Forum Message <> Reply to Message

Did you upload the latest version in svn?
Would be nice to have a benchmark!
Regards, Ion Lupascu(tojocky)

Mindtraveller wrote on Wed, 10 February 2010 11:290. Mt-Alt is more universal.
PostCallback expects callback to be called from main GUI thread. AFAIK, threads without GUI, and non-main threads don't have PostCallback and have no mechanism of queueing callbacks of their own.
Mt-Alt gives queue to any thread you want, including main/GUI and main/non-GUI thread.
So if you have a complex scenario of threads interaction (as I do), you may find useful the simplicity of doing this without any sync objects at all.

1. Mt-Alt is more lightweight.
This is due to avoid using U++ callbacks. In the early periods of Mt-Alt development, I've made a number of tests which finally led me to deny U++ callbacks for potentially-critical-bottleneck class like thread queue.
Each time you create callback, you create system core sync object, use it, and then delete it when callback is destroyed. More of that, operating system may have its own bounds for overall sync objects count.
So, if your system may have high loads and high number of callbacks int the queue, Mt-Alt is much more lightweight and fast.
Current project is being tested with high loads with ~100000 of callbacks in the queue, and it works well.

2. Mt-Alt is more safe and stable.
Everything you call through PostCallback is called in the application "idle" period. This means that if you give too much work for the main GUI thread (or you run your app on rather slow/old PC), it is never idle. And your "new" callbacks are executed with big lag or never executed at all.
The bigger problem connected to one above, is that if you use program for some amount of time, you may have too lengthy internal queue of callbacks. And I don't know if there are any safe ways to guarantee that PostCallback after 2-3 hours of hard CPU work doesn't create exception from callbacks container (old callbacks are still in the queue while you still add new ones).
Mt-Alt proposes two solutions here.
First, mechanism of "pushing old out" if your queue is too big. Also Mt-Alt has ability to call functions from main GUI loop, so you have more adequate behaviour on highly loaded apps and apps which could be installed for slow/old hardware.

I use U++ for hardware automation and operator interface which is commonly used on industrial PCs and even controllers. So I have to optimize and have to be shure that app works well after a pair of months of hard work without switching off or closing my app.