Subject: "Alternative Multithreading" revisited
Posted by Mindtraveller on Mon, 29 Jun 2009 19:00:45 GMT
View Forum Message <> Reply to Message

Finally I have finished some large U++ application and I have a number of new things tested while writing and debugging it. Currently each of them at release state and I guarantee (almost?) perfect work, although it would be great to have any feedback.

I`d like to introduce first of them here. It is alternative approach to multithreading which was discussed some months ago in this forum. What does it do? Actually it saves lot of your time and eliminates headache of synchronization objects.

Application mentioned above had about 5 threads which were interacting rather actively with each other. And during the months of debugging I didn`t have ANY problems with threads synchronization or something like locks or conflicts. Imagine: 5 threads and no problems!

Of course, this solution has it`s boundaries. It shouldn`t be applied in cases where you have extremely active interaction between threads (actually, more than 300-400 inter-threading "messages"/calls per second on my AMD 2 GHz starts to make difference). In any other cases it is OK to use it.

If you are still interested, let's discuss how to use proposed solution. First of all, it introduces a class/threading model:

Each thread is an object of CallbackThread class descendant
Main (GUI) thread is described as the object-descendant of CallbackQueue class (in non-GUI applications you still may have it)
There are NO public members (it is really OOP-style) and any interaction between threads are made through public member functions
Each thread (including main) has it`s internal queue of "callbacks" which are executed consequently in FIFO-order. All the functions are executed in object's thread.

That is why you don`t need synchronization objects. Threads interact with some public member functions (delegates/messages) only and they "know" nothing more about each other. So, thread's private functions are executed (handled) in it's own thread and don`t need to do anything with synchronization.

I've added two packages to see and test it. First, MtAlt is the main class to use alternative multithreading. MtAltExample1 is the simple example of it's usage.

Any questions or suggestions are welcome.

File Attachments

1) MtAlt.zip, downloaded 699 times

I'd like to see alternative-MT in the Bazaar(total votes: 13)


Yes 12/(92%)
No 1/(8%)