Subject: Re: Thoughts about alternative approach to multithreading
Posted by Mindtraveller on Mon, 26 Jan 2009 23:06:48 GMT
View Forum Message <> Reply to Message

Finally I`ve finished some controller programming work and resumed developing of "alternative MT-approach" class.
Let`s summarize the idea. To avoid synchronization objects headache and possible synchronization issues it is proposed to make all the threads isolated from each other. It means that
* threads do not "see" each other`s variables (including global ones, the ones with atomic access and shared synchronization objects) - it fully conforms classic OOP approach.
* threads do not call other thread`s member functions directly
* the only possible threads interaction is made through sending some message into thread`s internal queue.

Generally this would make threads interactions far more predictable and debuggable.
After some thinking I`ve made a decision not to send messages but instead send requests to run thread public member function (callback). It was decided because "message" is actually a signal to do some job. So it is no need to create any artificial message identifiers while you directly request to execute these callbacks.

It looks like this:

```
class JobThread1 : public CallbackThread
{
public: //these members may be added into thread queue and must not be called directly
   void Job11();
   void Job12(const String &);
private: //all the realization details are private
   //...
};

class class JobThread2 : public CallbackThread
{
public: //these members may be added into thread queue and must not be called directly
   void Job21(const String &);
private: //all the realization details are private
   //...
};

CONSOLE_APP_MAIN
{
   JobThread1 jobs1;
   JobThread2 jobs2;

   for (int i=0; i<10; ++i)
   {
      jobs1.Add(&JobThread1::Job11);
      jobs1.Add(&JobThread1::Job12, FormatInt(i));
```

```
        jobs2.Add(&JobThread2::Job21, FormatIntHex(i));
    }
};
```

You may even treat main thread as the same alt-MT thread. To do this you may have CallbackQueue variable and request it`s tasks. These tasks are processed with CallbackQueue::DoTasks().

Below is ready-to-use class with a simple test code.
If you are interested you may download and test it, or even use it in your apps. I hope more advanced versions of class will come soon (a number of optimizations is yet to be made).

"Alternative" MT requires a bit of reengineering threads processing functions. To make alt-MT program, you should think differently. Now you can`t share variables, now you don`t have a number of routines which are called in unpredictable sequence.

Instead you have messaging queues which mustn`t hold big number of messages (it is possible though not recommended). This means that thread must be logically solid as much as possible. This would keep all "tiny" interactions inside one thread.

Of course there are situations where "classic" MT with sync objects fits better. Situations include threads exchange with a huge number of tiny messages (inc/dec some variable). So if you cannot divide threads interaction into a (relatively small) number of (relatively medium) jobs - use "classic" MT.

I mean if you use more than 100`000 of messages per second - just use sync objects instead. But now you at least may choose to think "new way" on reorganizing threads member functions to make program stable or continue interacting with shared global variables to make it possibly quicker and less memory consuming but harder to debug and potentially less stable.

File Attachments

1) TestEtude2.zip, downloaded 367 times