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 657 times

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

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

Subject: Re: "Alternative Multithreading" revisited
Posted by gridem on Mon, 29 Jun 2009 19:51:15 GMT
View Forum Message <> Reply to Message

The idea is very interesting. But the attached example isn't compiled on Linux platform:

----- CtrlLib (GUI MT GCC DEBUG SHARED DEBUG_FULL BLITZ LINUX POSIX) (1 / 10)
----- MtAlt (GUI MT GCC DEBUG SHARED DEBUG_FULL BLITZ LINUX POSIX) (2 / 10)
WARNING: /home/grigory/MyApps/MtAlt/MtAlt.cpp has invalid (future) time 06/29/2009 18:32:40
WARNING: /home/grigory/MyApps/MtAlt/MtAlt.h has invalid (future) time 06/29/2009 20:04:58
MtAlt.cpp
/home/grigory/MyApps/MtAlt/MtAlt.cpp: In function 'Upp::dword rdtsc()':
/home/grigory/MyApps/MtAlt/MtAlt.cpp:9: error: expected `(' before '{' token
/home/grigory/MyApps/MtAlt/MtAlt.cpp:11: error: expected `;' before 'mov'

*** 2 errors, 0 warnings
MtAlt: 1 file(s) built in (0:04.69), 4692 msecs / file, duration = 4723 msecs

There were errors. (0:06.22)

And one more question:
Why do you use the own callback system instead of Upp callbacks?

Thank you for your tempting MT GUI approach.

Subject: Re: "Alternative Multithreading" revisited
Posted by mirek on Mon, 29 Jun 2009 20:15:40 GMT
View Forum Message <> Reply to Message

Mindtraveller wrote on Mon, 29 June 2009 15:00

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.

Well, I have some experiences now (did project based on queues, now planning to rewrite it to plain old locking) and I have something to say about the topic (IMO!):

Synchronization objects are simple to manage as compared to often complex race condition relations in queued systems.

IMO, this is the exactly same problem that seems to have killed microkernels.

Mirek

Subject: Re: "Alternative Multithreading" revisited
Posted by Mindtraveller on Mon, 29 Jun 2009 21:15:46 GMT
View Forum Message <> Reply to Message

gridem wrote on Mon, 29 June 2009 23:511. But the attached example isn't compiled on Linux platform
2. Why do you use the own callback system instead of Upp callbacks?
1. Please try to download archive again from the first post (I`ve reuploaded corrected version) and rebuild.
2. This is good question. I don`t use U++ callbacks because each U++ callback uses system`s synchronization object (link). This makes too big drawback in efficiency in my case. So I use my own implementation, rather small but less universal comparing to U++ ones.

Subject: Re: "Alternative Multithreading" revisited
Posted by Mindtraveller on Mon, 29 Jun 2009 21:53:10 GMT
View Forum Message <> Reply to Message

luzr wrote on Tue, 30 June 2009 00:151. did project based on queues, now planning to rewrite it to plain old locking
2. Synchronization objects are simple to manage as compared to often complex race condition relations in queued systems.
3. IMO, this is the exactly same problem that seems to have killed microkernels.
1. Mirek, this is very interesting why you decided to switch back to "classic-MT". Could we discuss it in the old topic? I've written that this approach has it`s boundaries, and it could be you jast had that case.
2. As I remember debugging app with 3+ threads and classic sync. objects - this was complete hell.
3. This is interesting too and I'd like to discuss it in the same topic. Microkernel is certainly very specific program with very strong requirements in efficiency, memory and overhead. Of course it should handle hundreds of thousands of messages per second. Alternative-MT is closer to classic GUI apps where you shouldn`t quickly handle large masses of data between threads. The same is for some types of internet server applications. In the other hand, most typical applications use 10-100 events per second and it is OK for alt-MT.

Subject: Re: "Alternative Multithreading" revisited
Posted by Mindtraveller on Tue, 09 Feb 2010 22:37:58 GMT

View Forum Message <> Reply to Message

Did anyone try to use MTAlt? Or it is useless for everyone? Or should I post detailed help about these classes?

P.S. I'd like to update a file in the SVN: /bazaar/MtAlt/MtAlt.h. Please do it someone who has access to bazaar sources.

File Attachments
1) MtAlt.h, downloaded 519 times

Subject: Re: "Alternative Multithreading" revisited
Posted by koldo on Wed, 10 Feb 2010 08:29:43 GMT
View Forum Message <> Reply to Message

Hello Mindtraveller

Now I use Upp MT using PostCallback(). From user side I understand how to use it and it works well in MinGW. It lets tenths of threads acting simultaneously over the Gui in the main thread.

From this point of view, what are the advantages of Alt-MT ?

Subject: Re: "Alternative Multithreading" revisited
Posted by Mindtraveller on Wed, 10 Feb 2010 09:29:13 GMT
View Forum Message <> Reply to Message

0. 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.

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.

Subject: Re: "Alternative Multithreading" revisited
Posted by koldo on Thu, 11 Feb 2010 07:15:21 GMT
View Forum Message <> Reply to Message

Mindtraveller wrote on Tue, 09 February 2010 23:37Did anyone try to use MTAlt? Or it is useless for everyone? Or should I post detailed help about these classes?

P.S. I'd like to update a file in the SVN: /bazaar/MtAlt/MtAlt.h. Please do it someone who has access to bazaar sources.
Hello Pavel

If you think your work is useful, follow developing it, improving the example and adding help.

Sometimes people is shy and do not give you feedback so you finally think nobody is using it.

Subject: Re: "Alternative Multithreading" revisited
Posted by Mindtraveller on Fri, 21 May 2010 21:23:32 GMT
View Forum Message <> Reply to Message

Please upload new version of MtAlt.

Added support for fast 5-argument callbacks (copy and pick version).

Planned: Priority thread queue. (Mirek, I've invented how to evade explicit Mutex / second queue issue - if you remember our discussion).

File Attachments

1) MtAlt.zip, downloaded 534 times

Subject: Re: "Alternative Multithreading" revisited
Posted by Mindtraveller on Thu, 21 Apr 2011 12:29:19 GMT
View Forum Message <> Reply to Message

I guess noone is using CallbackThread classes except me. May be it is good idea to remove this package from official Bazaar.

Subject: Re: "Alternative Multithreading" revisited
Posted by zsolt on Thu, 21 Apr 2011 14:35:49 GMT
View Forum Message <> Reply to Message

Mindtraveller wrote on Thu, 21 April 2011 14:29May be it is good idea to remove this package from official Bazaar.
I don't think so. I have learned a lot from that. And it would be useful in embedded systems, I think.

Please, don't remove it.

Subject: Re: "Alternative Multithreading" revisited
Posted by koldo on Sun, 24 Apr 2011 16:31:23 GMT
View Forum Message <> Reply to Message

Hello Konstantin

I have the same opinion.

IMHO:
- if you think your package is valuable for U++ users (or at least is better than other packages inside Bazaar)

- and you will maintain it.

Bazaar will be glad to host your Package .