Subject: AsyncWork
Posted by mirek on Sat, 14 Oct 2017 09:34:45 GMT
View Forum Message <> Reply to Message

AsyncWork is U++ take on std::future mechanism. The difference is that AsynWork is using CoWork thread pool as backend and, more importantly, allows for cancelation of job.

```
#include <Core/Core.h>

using namespace Upp;

CONSOLE_APP_MAIN
{
 StdLogSetup(LOG_FILE|LOG_COUT);

 auto a = Async([](int n) -> double {
  double f = 1;
  for(int i = 2; i <= n; i++)
   f *= i;
  return f;
 }, 100); // Schedules job to be executed by threadpool, returns AsyncWork for the return value and job control

 DUMP(a.Get()); // Makes sure job is finished (can execute it if it has not started yet), returns the result

 auto b = Async([] { throw "error"; });

 try {
  b.Get(); // exception is propagated
 }
 catch(...) {
  LOG("Exception has been caught");
 }


 auto c = Async([] {
  for(;;)
   if(CoWork::IsCanceled()) {
    LOG("Work was canceled");
    break;
   }
 });
 Sleep(100); // make it chance to start
 // c destructor cancels the work (can be explicitly canceled by Cancel method too)
}
```

Subject: Re: AsyncWork
Posted by Oblivion on Sun, 15 Oct 2017 07:58:23 GMT
View Forum Message <> Reply to Message

Great work, thanks!

Best regards,
Oblivion

Subject: Re: AsyncWork
Posted by koldo on Sun, 15 Oct 2017 16:40:14 GMT
View Forum Message <> Reply to Message

Dear Mirek, dear Oblivion

For a standard GUI program to have a more responsive user interface, do you propose AsynWork to launch all jobs?
(And of course, sorry for this reference, to forget forever any Ctrl::ProcessEvents() call :( ).

Subject: Re: AsyncWork
Posted by mirek on Sun, 15 Oct 2017 17:00:54 GMT
View Forum Message <> Reply to Message

koldo wrote on Sun, 15 October 2017 18:40Dear Mirek, dear Oblivion

For a standard GUI program to have a more responsive user interface, do you propose AsynWork to launch all jobs?
(And of course, sorry for this reference, to forget forever any Ctrl::ProcessEvents() call :( ).

Definitely NO.

IME, for GUI, you need to use Thread(s).

Subject: Re: AsyncWork
Posted by koldo on Mon, 16 Oct 2017 07:04:51 GMT
View Forum Message <> Reply to Message

Thank you Mirek.

(Sorry Mirek and Oblivion for an out of topic question :roll:)
In your opinion, how is the best way from the Threads to interact with GUI?:
- GuiLock
- The GUI uses SetTimeCallback() to set a timer function that updates the GUI thanks to shared variables.

Subject: Re: AsyncWork
Posted by Oblivion on Mon, 16 Oct 2017 07:50:27 GMT
View Forum Message <> Reply to Message

Quote:Dear Mirek, dear Oblivion

For a standard GUI program to have a more responsive user interface, do you propose AsynWork to launch all jobs?
(And of course, sorry for this reference, to forget forever any Ctrl::ProcessEvents() call Sad).

Hello Koldo,

I agree with Mirek.
You should use dedicated Thread(s) (in contrast to worker threads).
While AsyncWork, hence CoWork, does a decent job here (considering that it has cancellation feature and -"experimental"- pipe support), disadvantages of thread pools in general comes into play in such scenarios.

I don't want to go into details here, so just to keep it short:

You'll usually have no control over the states and priorites of workers, or exact execution time, or sometimes, if the job will be executed in worker threads at all. Therefore building a UI on AsyncWork, or thread pools in general, is not a very good idea. UI or any other time consuming operation can easily take advantage of it, but IMO it should not be built upon it.
(See the new SSH package, for a concrete -experimental- example of AsyncWork, and how it makes things simpler (and faster, up to 2-4 times).)

P.s. But there is a middle ground if you really need something like that. An attempt build a hybrid of worker thread concept, and dedicated threads, with a reasonble control over the given thread's states and priority: Job (the one I wrote). You may also want to try that.

Best regards,
Oblivion.

Subject: Re: AsyncWork
Posted by mirek on Mon, 16 Oct 2017 08:29:14 GMT
View Forum Message <> Reply to Message

koldo wrote on Mon, 16 October 2017 09:04Thank you Mirek.

(Sorry Mirek and Oblivion for an out of topic question :roll:)
In your opinion, how is the best way from the Threads to interact with GUI?:
- GuiLock
- The GUI uses SetTimeCallback() to set a timer function that updates the GUI thanks to shared variables.

GuiLock is definitely better (it is a 'newer' thing too), except the cases where it cannot be used (that is where windows are created - this limitation is caused by win32 basic design, which creates 'per-thread'

SetTimeCallback is really just the 'oldest' mechanism. At that time, it was choosen as the 'safe' and straightforward way.

Also note that there is Ctrl::Call, which is 'synchronous' variant of SetTimeCallback. Basically performs the code in the main thread, but waits for its completion.

Now, practical experience: If all you want to do is to have application responsive while long operation is being performed, maybe with some fancy progress bar, all you need is GuiLock.

Mirek

Subject: Re: AsyncWork
Posted by koldo on Tue, 17 Oct 2017 06:55:56 GMT
View Forum Message <> Reply to Message

Thank you very much :)

In addition GuiLock is easier and safer than sharing variables.

It is just a matter of using it wisely.
For example, this would be forbidden:

```
EditInt edit;
int count = 0;
while(true) {
 count++;
 GuiLock __;
 edit <<= count;
}
```

In addition if a GUI action (pushing a button) launches a Thread, it would be necessary to disable the button to avoid pushing it again while Thread is running.

Subject: Re: AsyncWork
Posted by mirek on Tue, 17 Oct 2017 11:33:18 GMT
View Forum Message <> Reply to Message

koldo wrote on Tue, 17 October 2017 08:55Thank you very much :)

In addition GuiLock is easier and safer than sharing variables.

Actually, GuiLock is exactly about sharing variables - all global variables of GUI...

Quote:
For example, this would be forbidden:

```
EditInt edit;
int count = 0;
while(true) {
 count++;
 GuiLock __;
 edit <<= count;
}
```

Not quite sure why this should be forbidden. This is OK, from framework perspective. Of course, it would probably be weird if user tried to write something into the field in the same time, but it is legal and would work just as expected.

Quote:
In addition if a GUI action (pushing a button) launches a Thread, it would be necessary to disable the button to avoid pushing it again while Thread is running.

Definitely. But that is not U++ responsibility, that is about application design.

Mirek

Subject: Re: AsyncWork
Posted by koldo on Wed, 18 Oct 2017 06:54:57 GMT
View Forum Message <> Reply to Message

Perfect :)
