Subject: AsyncWork, IsFinished() may not be working properly
Posted by Oblivion on Wed, 28 Mar 2018 17:17:20 GMT
View Forum Message <> Reply to Message

Hello,

Either I am doing something wrong, or AsyncWork/CoWork's IsFinished() method is not working properly (Tested on latest Upp/GCC/Linux kernel).

It always returns true;

Here is a simple test code to show the problem:

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

using namespace Upp;

CONSOLE_APP_MAIN
{
 StdLogSetup(LOG_FILE|LOG_COUT);

 Array<AsyncWork<void>> workers;

 for(int i = 0; i < 4; i++) {
  workers.Add() = Async([=]{ LOG("Started, worker #" << i); for(;;); });
 }
 Sleep(5000);

// while(!workers.IsEmpty())
  for(int i = 0; i < workers.GetCount(); i++) {
   auto& w = workers[i];
   if(w.IsFinished())
    LOG("Stopped, worker #" << i); // <<- Called, when it shouldn't.
  }
}
```

In the above test code, workers are supposed to run forever. So IsFinished should (?) return false. But when I check them with IsFinished() method, I get positive results every time.

Am I doing something wrong?

Here is the log:

CoWork constructed 7f86caf82060
Do0, looper: false, previous todo: 0
Pool::InitThreads: 8
CoWork thread #0 started
CoWork thread #1 started
CoWork thread #2 started
CoWork thread #3 started
CoWork thread #4 started
CoWork thread #5 started
CoWork thread #6 started
CoWork thread #7 started
#1 Waiting for job
Adding job
Releasing thread waiting for job, waiting threads: 1
#1 Waiting ended
#1 Job acquired
DoJob (CoWork 7f86caf82060)
Started worker #0
#4 Waiting for job
CoWork constructed 7f86caf82140
Do0, looper: false, previous todo: 0
#5 Waiting for job
#0 Waiting for job
#3 Waiting for job
#6 Waiting for job
#7 Waiting for job
#2 Waiting for job
Adding job
Releasing thread waiting for job, waiting threads: 7
CoWork constructed 7f86caf82220
Do0, looper: false, previous todo: 0
#4 Waiting ended
#4 Job acquired
DoJob (CoWork 7f86caf82140)
Started worker #1
Adding job
Releasing thread waiting for job, waiting threads: 6
CoWork constructed 7f86caf82300
Do0, looper: false, previous todo: 0
Adding job
Releasing thread waiting for job, waiting threads: 6
#0 Waiting ended
#0 Job acquired
DoJob (CoWork 7f86caf82300)
Started worker #3
#5 Waiting ended
#5 Job acquired

DoJob (CoWork 7f86caf82220)
Started worker #2
Stopped worker #0
Stopped worker #1
Stopped worker #2
Stopped worker #3
CoWork Cancel0
WaitForFinish (CoWork 7f86caf82060)
//...

Best regards,
Oblivion

Subject: Re: AsyncWork, IsFinished() may not be working properly
Posted by mirek on Sat, 31 Mar 2018 05:49:28 GMT
View Forum Message <> Reply to Message

Hopefully fixed. (CoWork::IsFinished bug).

Final testcase:

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

using namespace Upp;

CONSOLE_APP_MAIN
{
 StdLogSetup(LOG_FILE|LOG_COUT);

 Array<AsyncWork<void>> workers;

 for(int i = 0; i < 4; i++) {
  workers.Add() = Async([=]{ LOG("Started, worker #" << i); Sleep(500); });
 }

 while(!workers.IsEmpty())
  for(int i = 0; i < workers.GetCount(); i++) {
   auto& w = workers[i];
   if(w.IsFinished()) {
    LOG("Stopped, worker #" << i); // <<- Called, when it shouldn't.
    workers.Remove(i);
    break;
   }
```

```
  }
}
```

Subject: Re: AsyncWork, IsFinished() may not be working properly
Posted by Oblivion on Sat, 31 Mar 2018 10:51:01 GMT
View Forum Message <> Reply to Message

Hello Mirek,

Thanks! Now it's working as expected.

Best regards,
Oblivion.

Subject: Re: AsyncWork, IsFinished() may not be working properly
Posted by JeyCi on Fri, 17 Jul 2020 10:48:50 GMT
View Forum Message <> Reply to Message

in U++v.13068 x32 using last example code in the topic I'm getting such result
Started, worker #3
Started, worker #2
Started, worker #0
Started, worker #1
Stopped, worker #3
Stopped, worker #2
Stopped, worker #0
Stopped, worker #0
? is it some form of race condition (last 2 lines when stopping workers)... how it can be corrected?

Subject: Re: AsyncWork, IsFinished() may not be working properly
Posted by JeyCi on Sat, 18 Jul 2020 05:51:27 GMT
View Forum Message <> Reply to Message

I found below code seems working good - without Race_Condition - (in u++ v.13068 windows x32)...

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

using namespace Upp;

CONSOLE_APP_MAIN
{
 StdLogSetup(LOG_FILE|LOG_COUT);
```

```
 Array<AsyncWork<int>> workers;

 // add worker
 for(int i = 0; i < 4; i++) {
  workers.Add() = Async([=]{ LOG("Started, worker #" << i); return i; });
 }

 Sleep(10);  //10/1000sec
 for (auto &fut : workers) {
  LOG("Stopped, worker #" << fut.Get());
 }
}
```

? BUT I wonder if there is a way to use atomic to counter to avoid race condition in the mirek's code?.. because any way I tried to use it (I mean Atomic in his code) - I have been still getting race condition :(

Subject: Re: AsyncWork, IsFinished() may not be working properly
Posted by Oblivion on Sat, 18 Jul 2020 07:59:07 GMT
View Forum Message <> Reply to Message

Hello JeyCi,

And welcome to the U++ forums!

Note that when the worker is removed from the array in the first example, the item count is also decreased by 1.
This obviously invalidates the indices, so the code breaks the for loop and starts it all over again (from 0).
Hence the "two or more zeros" in the log output. It might give the impression of race condition, but it is not. (If that's the problem.)

(The reason is that the first example was about a problem in IsFinished(). It was always returning true.
The example did not rely on those indices for its purpose, so I didn't care to make it more correct...)

Please try this one instead:

```
CONSOLE_APP_MAIN
{
 StdLogSetup(LOG_FILE|LOG_COUT);

 Array<AsyncWork<int>> workers;
```

```
 for(int i = 0; i < 4; i++) {
  workers.Add() = Async([=]{ LOG("Started, worker #" << i); Sleep(500); return i; });
 }

 while(!workers.IsEmpty())
  for(int i = 0; i < workers.GetCount(); i++) {
   auto& w = workers[i];
   if(w.IsFinished()) {
    LOG("Stopped, worker #" << w.Get());
    workers.Remove(i);
    break;
   }
  }
}
```

Quote:
I found below code seems working good

Yes, that works, because, unlike the "non-blokcking" first example, this one has a "blocking" loop:

```
 for (auto &fut : workers) {
  LOG("Stopped, worker #" << fut.Get());
 }
```

The Get() method will wait the worker to finish its job unless it is alreay finished. So it is "blocking".

Best regards,
Oblivion

Subject: Re: AsyncWork, IsFinished() may not be working properly
Posted by JeyCi on Sat, 18 Jul 2020 12:21:45 GMT
View Forum Message <> Reply to Message

Oblivion , thank you very much - you have got together some pieces of information I know but still lack experience in its use - you have pointed the sides of its practical application...
1. I already used the idea about .Remove(i) - thanks to U++ examples. But thanks for your explanation, why I was getting zeros in mirek'code - I didn't noticed this easy reason
2. you are right - it was really NOT a race condition yet
3. now I see that really my code is blocking, & mirek's non-blocking, and your example is blocking as well...

... Yet I'm a beginner at parallel programming - I considered non-blocking algorithm to be better than blocking... but now I understood - they are simply for different purposes... and if I need to return value to the main thread - of course I need blocking as synchronization should be done - I think so now...
I'm just having the hope - if there is a way to return from mirek's non-blocking example value e.g. with lock-free mechanism of atomic?... or in any case we will need synchronization (blocking) here to return value from each thread?..
P.S.
I think now - it was just a hope...
in any case thank you for increasing my practical understanding such fresh knowledge for me :)
Best regards,

Subject: Re: AsyncWork, IsFinished() may not be working properly
Posted by Oblivion on Sat, 18 Jul 2020 13:31:16 GMT
View Forum Message <> Reply to Message

Happy to help :)

Quote:if there is a way to return from mirek's non-blocking example value

Which example, exactly, are you referring to?

The above code (in which IsFinished() method is called) is already getting the value without any locking mechanism, and is non-blocking. IsFinished() method allows you to check the progress of a given worker instance without waiting on it (i.e. non-blocking). Thus, if the work is finished then Calling Get() won't wait either, it will immediately return the resultitng value of the offloaded computation.

Let me put it this way:

```
CONSOLE_APP_MAIN
{
 StdLogSetup(LOG_FILE|LOG_COUT);

 Array<AsyncWork<Vector<int>>> workers;
 Vector<int> results;

 for(int i = 0; i < 10; i++) {
  workers.Add() = Async([=]{
   Sleep(Random(200));
   Vector<int> v;
   for(int j = i * 10; j < i * 10 + 10; j++)
    v.Add(j);
   return pick(v);
  });
 }
```

```
 while(!workers.IsEmpty())
  for(int i = 0; i < workers.GetCount(); i++) {
   auto& w = workers[i];
   if(w.IsFinished()) { // Non blocking check. It will not wait.
    // No locking required, since we dont use a shared variable...
    auto iota = w.Pick(); // Same as Get() but picks ("moves") the resulting vector.
    RDUMP(iota);
    results.Append(iota);
    Sort(results);
    RLOG("Sorted partial results: " << results);
    workers.Remove(i);
    break;
   }
  }

 RLOG("Final results: " << results);
}
```

Hopefully, this example can demonstrate the basic usage.

If you want a "real" example, check the SshBasics/SftpMT.cpp example here:
https://www.ultimatepp.org/reference$SshBasics$en-us.html

IF you have more questions and/or need more help, let me know.

Best regards,
Oblivion

Subject: Re: AsyncWork, IsFinished() may not be working properly
Posted by JeyCi on Sat, 18 Jul 2020 14:44:38 GMT
View Forum Message <> Reply to Message

Oblivion wrote on Sat, 18 July 2020 15:31if the work is finished then Calling Get() won't wait either, it will immediately return the resultitng value of the offloaded computation.
Yes, now I see - this is important difference in your code comparing it with mine
and here
Oblivion wrote on Sat, 18 July 2020 15:31while(!workers.IsEmpty())
  for(int i = 0; i < workers.GetCount(); i++)
? is it duplication of loop & what for? why not use only one any row from these 2, not both?

Oblivion wrote on Sat, 18 July 2020 15:31// No locking required, since we dont use a shared variable...
- well, I worried for a long time about this i - always doubting if it is shared variable - it always

seems to me that counter itself is always shared & should be locked somehow when making i++... it's difficult to understand why not?..
Oblivion wrote on Sat, 18 July 2020 15:31auto iota = w.Pick(); // Same as Get() but picks ("moves") the resulting vector.
- I liked this row :)

Subject: Re: AsyncWork, IsFinished() may not be working properly
Posted by JeyCi on Sat, 18 Jul 2020 15:06:06 GMT
View Forum Message <> Reply to Message

JeyCi wrote on Sat, 18 July 2020 16:44 while(!workers.IsEmpty())
  for(int i = 0; i < workers.GetCount(); i++) {
? is it duplication of loop & what for? why not use only one any row from these 2, not both?
sorry to hurry with my question - I understood -
1 cycle "for" from the beginning till the end of the container;
second cycle (while) - to repeat again & again for-cycle from the beginning till the end of the container - until container IsEmpty... because after first for-loop some worker-threads can be still not finished... really logically

JeyCi wrote on Sat, 18 July 2020 16:44
- it always seems to me that counter itself is always shared & should be locked somehow when making i++...

but frankly speaking, if you insist that not, I can agree with you now... because this i (counter) is still in one (primary) thread, from where it is taken to workers - Geniusly... thanks a lot
===
after your example - more clear is now my view at threads...

Subject: Re: AsyncWork, IsFinished() may not be working properly
Posted by Oblivion on Sat, 18 Jul 2020 15:55:20 GMT
View Forum Message <> Reply to Message

Quote:1 cycle "for" from the beginning till the end of the container;
second cycle (while) - to repeat again & again for-cycle from the beginning till the end of the container - until container IsEmpty... because after first for-loop some worker-threads can be still not finished

Exactly.

But indeed we could have written it using a single loop. However, I would preferer not to. My reasons:

1) It increases code compexity, even if by a small margin. When it becomes a habit, it will make it a pain to read and debug even your own code. :)

2) This is an example, and I prefer examples displaying their logic as clearly as possible. (Although I admit I have a long way to go in this aspect...)

Quote:
- it always seems to me that counter itself is always shared & should be locked somehow when making i++...
but frankly speaking, if you insist that not, I can agree with you now... because this i (counter) is still in one (primary) thread, from where it is taken to workers

Well, in genral, this is the rule to follow, yes. But not in this case, no.
The lambda expressions/functions support is arguably one of the best thing that happened to C++.
If you look closely, you can see that we are capturing the index (i) "by value" (i.e. we are copying it):

[=] -> lambda: capture all by value (basically, copies or picks the local variables, and the implicit "this" pointer if it is called from a class/struct.)
[&] -> lambda: capture all by reference (passes a reference) This can be quite dangerous, depending on how it is used.

Thus, in our code we are iterating the index in a for loop, and we copy each iteration to a thread. w0 (worker 0) gets i = 0, w1 -> i=1, w2 -> i=2, and so on...

It's a long topic to cover here, so I suggest you reading C++ lambda functions and capture types, for more details, specifically Scott Meyers' famous book "Effective Modern C++". It is one of my all-time favorites.

:)

Best regards,
Oblivion

Subject: Re: AsyncWork, IsFinished() may not be working properly
Posted by JeyCi on Sat, 18 Jul 2020 16:34:52 GMT
View Forum Message <> Reply to Message

Oblivion wrote on Sat, 18 July 2020 17:55
If you look closely, you can see that we are capturing the index (i) "by value" (i.e. we are copying it):

I think I see now in the class AsyncWork in the library - something like this

```
void	Do(Function&& f, Args&&... args) { co.Do([=]() { ret = f(args...); }); }
```

- thus, by value - ok I will know... thanks!
p.s.
but in my U++ v.13068 mingw win-x32 I have no .Pick() to test your last code... (please don't ask me to update the u++ version - because I have win-x32 & last for it compiled 13664 just is having some problems with one code, about which I yet need some tests to do)... But if some changes can be done to your last code - to start it working without .Pick?.. because am trying to use .Get, neither = (deleted function), nor auto& iota (reference) helps...

Subject: Re: AsyncWork, IsFinished() may not be working properly
Posted by Oblivion on Sat, 18 Jul 2020 17:24:23 GMT
View Forum Message <> Reply to Message

Quote:- thus, by value - ok I will know... thanks!

No, that is an implementation detail. You shouldn't worry about that.
What I've meant is (in the test code above):

```
 for(int i = 0; i < 10; i++) {
  workers.Add() = Async([=]{ // <--- The capture point/syntax you should be aware of.
   Sleep(Random(200));
   Vector<int> v;
   for(int j = i * 10; j < i * 10 + 10; j++)
    v.Add(j);
   return v;
  });
 }
```

Quote:
(please don't ask me to update the u++ version - because I have win-x32 & last for it compiled 13664 just is having some problems with one code, about which I yet need some tests to do)... But if some changes can be done to your last code - to start it working without .Pick?.. because am trying to use .Get, neither = (deleted function), nor auto& iota (reference) helps...

Well, AFAIK, that's (picking without Pick() method) not possible unless you do some tricks, which will make things even more complicated.

My suggestion would be to patch the AsyncWork code to your local source tree (its 20-30 lines of code in a header file, thats all). Only two or three lines have to be patched.

Or

Again, copy the code, and rename it as AsyncWork2 (or whatever you prefer) and add it to your apps header:

The code you have to copy is in uppsrc/Core/CoWork.h :

```
template <class Ret>
class AsyncWork {
 template <class Ret2>
 struct Imp {
  CoWork co;
  Ret2   ret;

  template<class Function, class... Args>
  void       Do(Function&& f, Args&&... args) { co.Do([=]() { ret = f(args...); }); }
  const Ret2& Get()                      { return ret; }
  Ret2       Pick()                      { return pick(ret); }
 };

 struct ImpVoid {
  CoWork co;

  template<class Function, class... Args>
  void       Do(Function&& f, Args&&... args) { co.Do([=]() { f(args...); }); }
  void       Get()                       {}
  void       Pick()                      {}
 };

 using ImpType = typename std::conditional<std::is_void<Ret>::value, ImpVoid, Imp<Ret>>::type;

 One<ImpType> imp;

public:
 template< class Function, class... Args>
 void  Do(Function&& f, Args&&... args)         { imp.Create().Do(f, args...); }

 void       Cancel()                     { if(imp) imp->co.Cancel(); }
 static bool IsCanceled()                    { return CoWork::IsCanceled(); }
 bool       IsFinished()                 { return imp && imp->co.IsFinished(); }
 Ret        Get()                        { ASSERT(imp); imp->co.Finish(); return imp->Get(); }
 Ret        operator~()                  { return Get(); }
 Ret        Pick()                       { ASSERT(imp); imp->co.Finish(); return imp->Pick(); }

 AsyncWork& operator=(AsyncWork&&) = default;
 AsyncWork(AsyncWork&&) = default;

 AsyncWork()                             {}
```

```
 ~AsyncWork()                        { if(imp) imp->co.Cancel(); }
};

template< class Function, class... Args>
AsyncWork<
 typename std::result_of<
  typename std::decay<Function>::type
   (typename std::decay<Args>::type...)
 >::type
>
Async(Function&& f, Args&&... args)
{
 AsyncWork<
  typename std::result_of<
   typename std::decay<Function>::type
    (typename std::decay<Args>::type...)
  >::type
 > h;
 h.Do(f, args...);
 return pick(h);
}
```

This is all. :)

Or if you have some specific, simple example of your code, post it here, and let's see what we can come up with. :)
Best regards,
Oblivion

Subject: Re: AsyncWork, IsFinished() may not be working properly
Posted by JeyCi on Sat, 18 Jul 2020 19:02:40 GMT
View Forum Message <> Reply to Message

Oblivion wrote on Sat, 18 July 2020 19:24
workers.Add() = Async([=]{ // <--- The capture point/syntax you should be aware of.

oh :) I was so blind
Oblivion wrote on Sat, 18 July 2020 19:24
My suggestion would be to patch the AsyncWork code to your local source tree (its 20-30 lines of code in a header file, thats all). Only two or three lines have to be patched.
This is all.

perfectly - you have updated my 13068 with 30 lines & no need to wait new version for x32win :)
thank you very much - it really works (your code is ok)
Oblivion wrote on Sat, 18 July 2020 19:24
Or if you have some specific, simple example of your code, post it here, and let's see what we can come up with.

it will be real offtop here,
Toggle Spoilerbut if you're interested about the code - one person gave me an example of how to use from another class progress of GUI at progress-reference... in 13664 there are some problems with that code - also concerning CoWork, I think...
but I was just interested about using GUIprogress FROM another class () or even another thread I will need (I started from simple GUI class with 1 working-thread & my loading purposes) - & it was the answer how I can make decomposition of my code... so I've just started to practice C++ in U++ framework - I liked its user-friendly interface & already built portable versions & already created my little utility for my needed goals - U++ is really very convenient & now I see in your answers - with very ! good support from its community...
I know - CoWork changed since 13068...
but I'm now trying to refactor my quickly written & working code according to OOP - as a part of getting acquainted with new language in free time - really interesting...
and classes of U++ really are giving speed for any application developing (I never have deal with async & threads before) - thanks a lot!
I'm now updated :)