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