Subject: Re: RE: Job package: A scope-bound worker thread for non-blocking operations.
Posted by Oblivion on Tue, 10 Oct 2017 21:48:05 GMT
View Forum Message <> Reply to Message

Hello Mirek,

tested Async/AsyncWork with MSC 2017, MinGW on windows, and with GCC on linux.

- It compiles on MSC without any hiccup.
- It does not compile on GCC (7.2) or MinGW unless the nested classes are moved out, (That's why I wrote my prototype that way.) and "Ret" is changed to some other parameter name. Here are the error codes I get:

```
\AsyncTest.cpp (8): error: declaration of template parameter 'Ret' shadows template parameter
\AsyncTest.cpp (18): error: explicit specialization in non-namespace scope 'class
AsyncWork<Ret>'
\AsyncTest.cpp (19): error: template parameters not deducible in partial specialization:
\AsyncTest.cpp (31): error: too many template-parameter-lists
\AsyncTest.cpp (47): error: 'class AsyncWork<const char*>' has no member named 'Do' ():
 h. D o (f, args...);
\AsyncTest.cpp: In instantiation of 'AsyncWork<typename std::result_of<typename
std::decay<_Tp>::type(std::decay_t<Args>...)>::type> Async(Function&&, Args&& ...) [wit
h Function = ConsoleMainFn_()::<lambda(int)>; Args = {int}; typename std::result_of<typename
std::decay<_Tp>::type(std::decay_t<Args>...)>::type = int]':
\AsyncTest.cpp (56): required from here
\AsyncTest.cpp (47): error: 'class AsyncWork<int>' has no member named 'Do'
\AsyncTest.cpp (47): error: 'class AsyncWork<void>' has no member named 'Do'
\AsyncTest.cpp (11): error: 'AsyncWork<Ret>::Imp<Ret>::ret' has incomplete type
\AsyncTest.cpp (11): error: invalid use of 'void'
\AsyncTest.cpp (15): error: forming reference to void
\AsyncTest.cpp (34): error: 'struct AsyncWork<void>::Imp<void>' has no member named 'Get'; did
you mean 'ret'?
\AsyncTest.cpp (34): error: return-statement with a value, in function returning 'void' [-fpermissive]
```

- More importantly there seems to be something wrong with the exception propagation mechanism. For,

1) Sometimes it fails to catch the exception, and the application crashes with that exception.
2) When it catches the exception the application hangs at the end (after the "exception caught" message is printed.)
3) Sometimes the application simply hangs.

I got this erratic behaviour both on windows and on linux, on a single machine, so it maybe a local hardware problem, I need to investigate it further...

Quote:s a tricky catch with IsFinished:

```
template <class Range>
ValueTypeOf<Range> ASum(const Range& rng, const ValueTypeOf<Range>& zero)
{
 int n = rng.GetCount();
 if(n == 1)
  return rng[0];
 if(n == 0)
  return 0;
 auto l = Async([&] { return ASum(SubRange(rng, 0, n / 2)); });
 auto r = Async([&] { return ASum(SubRange(rng, n / 2, n - n / 2)); });
 while(!l.IsFinished() || !r.IsFinished())
  Sleep(1);
 return l.Get() + r.Get();
}
```

What do you think is wrong with this code? Smile

Mirek

Sure, but can this really be attributed to a design flaw?
I mean, f I'm not really missing anything else, it seems that here we simply have a careless programming.
Recursion is potentially tricky by nature, and requires the developer to be extra cautious with his/her assumptions.

A proper use of IsFinished() can be found in my JobBenchmark example , where it is simply used to check the worker, and move on to others if the job is not finished... (at least, that's what I have in my mind in the first place)

Best regards.
Oblivion