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