Subject: The right way to use CoDo with GuiLock?
Posted by Mountacir on Sat, 11 Jun 2022 10:25:15 GMT
View Forum Message <> Reply to Message

Hi,

I'm having trouble trying to update a ColumnList while using CoDo.
Adding GuiLock in my example makes the app freeze.
How to do it the correct way?

```
void DoCoTest::FillList(){
 std::atomic<int> ii(0);
 CoDo([&] {
  for(int i =ii++; i < 100 ; i=ii++) {

   GuiLock __;
   columnlist.Add(i);

  }

 });
}
```

Thank you.

Subject: Re: The right way to use CoDo with GuiLock?
Posted by Oblivion on Sat, 11 Jun 2022 12:43:42 GMT
View Forum Message <> Reply to Message

Hello Mountacir,

There can be several approaches here, but I'd suggest these two:

```
void CoTest::CoForTest1()
{
 list.Clear();
 Vector<int> v;
 CoFor(100, [=, &v](int i) {
  CoWork::FinLock();
  v.Add(i);
 });
 std::for_each(v.begin(), v.end(), [=](auto n) { list.Add(n); });
}
```

```
void CoTest::CoForTest2()
{
 list.Clear();
 GuiUnlock __; // Unlock the global Gui Mutex so it can be acquired by the workers...
 CoFor(100, [=](int i) {
  GuiLock __; // Acquire ...
  list.Add(i);
 });
}
```

CoFor is a convenience function that does what you just do in your example: iteration.

Both CoDo and CoFor are blocking functions. They will wait for the workers to join.

So, first you will need to release the gui lock of main gui thread, using the GuiUnlock class.

IMO, however, CoForTest1() would be a better and safer way, as the CoForTest2 will start blocking the GUI when the item count is high (try with N=10000 and see the difference in the results).

P.s:

This is your example, done in the right way:

```
void CoTest::CoDoTest()
{
 list.Clear();
 std::atomic<int> ii(0);
 GuiUnlock __;
 CoDo([=, &ii] {
  GuiLock __;
  for(int i =ii++; i < 100 ; i=ii++) {
   list.Add(i);

  }
 });
}
```

Best regards,
Oblivion

Subject: Re: The right way to use CoDo with GuiLock?
Posted by Mountacir on Sat, 11 Jun 2022 22:01:09 GMT
View Forum Message <> Reply to Message

Thank you so much Oblivion! You saved me from frustration.

I've tried something like CoForTest1() with CoDo, it was my only option.

I really appreciate taking time to explain to me, Thank you!!

Subject: Re: The right way to use CoDo with GuiLock?
Posted by Mountacir on Sun, 12 Jun 2022 14:40:45 GMT
View Forum Message <> Reply to Message

One more question please.

Is there any approach to CoWork for updating the ColumnList as every worker finish? instead of waiting till all of them are finished to update.

```
void CoTest::CoForTest2()
{
 list.Clear();

 for(int i = 0 ;i < 100 ;++i )
  list.Add("Calculating");

 GuiUnlock __;
 CoFor(100, [=](int i) {

  //Some Calculation

  GuiLock __;
  list.Set(i,"Result"); //How to make this show now?
 });
}
```

Thank you!

Subject: Re: The right way to use CoDo with GuiLock?
Posted by Oblivion on Sun, 12 Jun 2022 16:12:30 GMT
View Forum Message <> Reply to Message

Quote:
Is there any approach to CoWork for updating the ColumnList as every worker finish? instead of waiting till all of them are finished to update.

Not with CoDo or CoFor, no. (Not that I know of, at least)

The reason is, you'll need a non-blocking way so that the GUI can be refreshed, and events are processed.
Fortunately, there are several ways to do that.

But since you want a loop paralellization, you can do it using the CoWork::Do0 // Which is called by CoDo, anyway.

Here's how:

```
void CoTest::CoForTest2()
{
 list.Clear();
 CoWork co;

 std::atomic<int> ii(0);

 co.Do0([=, &ii] { // This is non blocking.. (i.e. will not wait for workers to compete their jobs)
  for(int i = ii++; i < 10000; i = ii++) {
   GuiLock __;
   list.Add(i);
  }
 },
 true);

 while(!co.IsFinished()) { // We'll use pseudo-blocking, so that we'll have control over the GUI event loop.
  ProcessEvents(); // Process the events.
  GuiSleep(20);    // Sleep right amount of time.
 }

}
```

Best regards,
Oblivion

Subject: Re: The right way to use CoDo with GuiLock?
Posted by Mountacir on Sun, 12 Jun 2022 18:05:24 GMT
View Forum Message <> Reply to Message

Quote:But since you want a loop paralellization, you can do it using the CoWork::Do0 // Which is called by CoDo, anyway.

The main reason I choose CoWork is that, it is straight forward to use. It creates the threads pool and schedule jobs.

Thank you very much Oblivion for your solution and explanation, I have way better understanding now.