Subject: Question about PostCallback from Child Thread
Posted by kfeng on Wed, 11 Jul 2007 14:12:02 GMT
View Forum Message <> Reply to Message

Hi,

I have a question about PostCallback.

Let's say I call the function from a child thread (parent thread is the main GUI loop):

```
  PostCallback(callback2(myInst, &MyClass::MyCB, arg1, arg2 ));
```

Is this a blocking call (synch) or non-blocking (asynch)?
I am asking, because if it's blocking, the result should be the same as:

```
PostCallback(callback(myInst, &MyClass:MyCB2));
void
MyClass::MyCB2()
{
  arg1 = GetArg1();
  arg2 = GetArg2();
}
```

My child thread writes to arg1 and arg2 - the GUI thread reads. If PostCallback doesn't block, I can have a race condition in the second case. Asynch would also imply that PostCallback() does some kind of locking of arg1 and arg2, so the child thread can't write to it while the GUI is reading.

I'm asking because what if I need to pass a bunch (10?) of arguments. Should I put them in 1 struct/class and use callback1(). I was lazy earlier today and tried the 2nd example above, but now, I'm having second thoughts...

Can someone explain the mechanism here?
Thank you.

Regards,
Ken

Subject: Re: Question about PostCallback from Child Thread
Posted by mirek on Wed, 11 Jul 2007 21:12:29 GMT
View Forum Message <> Reply to Message

Async / Non-blocking. It gets performed when GUI thread goes idle after processing messages (if any).

Callback queue itself is MT protected.

Also, callback gets invoked by GUI thread - it is in fact the best way how worker thread is supposed to communicate with it, means you do not need to worry too much about protecting data, there can only be a single GUI thread, means now two Callbacks from Callback (timer) queue can be processed at the same time.

And yes, to pass 10 parameters, put them into struct and use callback1.

Subject: Re: Question about PostCallback from Child Thread
Posted by kfeng on Wed, 11 Jul 2007 22:52:15 GMT
View Forum Message <> Reply to Message

Excellent. Thank you for the quick reply, Mirek.

Regards,
Ken

Subject: Re: Question about PostCallback from Child Thread
Posted by kfeng on Thu, 12 Jul 2007 04:11:39 GMT
View Forum Message <> Reply to Message

OK, suppose I have a C struct with a bunch of pointers to the heap:

```
struct
{
  int    *intP;
  double *doubleP;
  char   **strP;
...
}
```

It's filled out by the child process and I pass a pointer to an instance to PostCallback(). Will PostCallback() be smart enough to lock all the pointed-to members? If not, is there a way to make the child thread block and wait until the parent is done reading?

The problem is **strP - I need this to run fast so I don't want to be looping through the members locking each one by hand - may be simpler to just get the child to wait for the parent to finish. Is there a way I can do this to a child thread?

Thanks in advance.

Regards,

Ken

Subject: Re: Question about PostCallback from Child Thread
Posted by mirek on Thu, 12 Jul 2007 19:54:49 GMT
View Forum Message <> Reply to Message

kfeng wrote on Thu, 12 July 2007 00:11OK, suppose I have a C struct with a bunch of pointers to the heap:

```
struct
{
  int    *intP;
  double *doubleP;
  char   **strP;
...
}
```

It's filled out by the child process and I pass a pointer to an instance to PostCallback(). Will PostCallback() be smart enough to lock all the pointed-to members? If not, is there a way to make the child thread block and wait until the parent is done reading?

The problem is **strP - I need this to run fast so I don't want to be looping through the members locking each one by hand - may be simpler to just get the child to wait for the parent to finish. Is there a way I can do this to a child thread?

Now I am a little bit confused.... First of all, "pointers to the heap" sounds like a bad practice if you are following U++ path of things.... Anyway, surely can happen.

Then the question is what "locking each one by hand" is supposed to mean.

In MT, what you have to lock (using mutex) is data that at specific moment can be accessible by more than single thread. Is this the case? Note that if you e.g. create heap data than are only passed using PostCallback and are not references anywhere else, you do not need to lock.

Now another question is why do you want to lock one by one? You can have single mutex protecting the whole array.... and lock just before the loop, unlock after.

Mirek

Subject: Re: Question about PostCallback from Child Thread
Posted by kfeng on Fri, 13 Jul 2007 15:19:09 GMT

luzr wrote on Thu, 12 July 2007 21:54kfeng wrote on Thu, 12 July 2007 00:11OK, suppose I have a C struct with a bunch of pointers to the heap:

```
struct
{
  int    *intP;
  double *doubleP;
  char   **strP;
...
}
```

It's filled out by the child process and I pass a pointer to an instance to PostCallback(). Will PostCallback() be smart enough to lock all the pointed-to members? If not, is there a way to make the child thread block and wait until the parent is done reading?

The problem is **strP - I need this to run fast so I don't want to be looping through the members locking each one by hand - may be simpler to just get the child to wait for the parent to finish. Is there a way I can do this to a child thread?

Now I am a little bit confused.... First of all, "pointers to the heap" sounds like a bad practice if you are following U++ path of things.... Anyway, surely can happen.

Then the question is what "locking each one by hand" is supposed to mean.

In MT, what you have to lock (using mutex) is data that at specific moment can be accessible by more than single thread. Is this the case? Note that if you e.g. create heap data than are only passed using PostCallback and are not references anywhere else, you do not need to lock.

Now another question is why do you want to lock one by one? You can have single mutex protecting the whole array.... and lock just before the loop, unlock after.

Mirek

OK. My apologies. Let me start from the beginning. I am using a 3rd party C library for retrieving real-time financial tickdata. In my first non-GUI pure-C non-MT program. the sequence looks like this:

1. Connect to server
2. Blocking wait for the server to send tickdata
3. Read the tickdata
4. Release the memory using their library function (since they did the allocation, they need me to release when I'm done)
5. Goto 2

Now the ugly part is the data. It doesn't come in one big block. Instead it looks like this:

```
typedef struct _x {
  int nCols;
  int nRows;
  union {
    double* pD;
    int*    pI;
    char**  pS;
    ...
  }
} *xArray;
```

So you can imagine that xArray is an array of columns represented by pointers to different types. In other words, xArray represents a "table" as an array of individually contiguous chunks of memory. xArray[0].pI[0] represents the first integer is the first column, if the first column is known to be an integer type. xArray[0].pI[3] represents the fourth integer in the first column. Hence, we know that xArray[0].pI is one contiguous chunk of memory. It gets ugly if we have pS, I think. xArray[1].pS[1] is the second string in the second column which is a contiguous list of strings. As I understand it, locking xArray[0].pI is sufficient, but I must lock each of xArray[1].pS[0..(nRows-1)].

But maybe this is where my thinking is wrong. My confusion lies in the wrong assumption that I needed to lock all the little malloc'ed pieces of memory. After rethinking the problem, and rewriting this e-mail over many interations, I can understand your confusion (because >I< was confused!).

A. xArray result = ReadFromLibrary_Blocking(); // Child Thread
B. call PostCallback(callback(..., result); // Asynchronous in Child Thread
C. Goto A. // Child Thread

X. Read result into GUI // Parent GUI Thread
Y. Free result // Parent GUI Thread
Z. Automatically unlocks result on exit from method // Parent

Here are the facts:
* Child Thread creates new result for every A.
* Parent won't read result until GUI is ready and result is in the PostCallback() queue.
* Parent will free result when it's done and Child never cares because once it's passed into the parent, it never reads/writes to the variable again.

So what is pointed to by result will never be read/written by two different threads. Information flows in one direction, sequentially from child thread(s) to parent GUI thread, so in fact, it's >NOT< the "locking" service that is required from PostCallback - rather, it's the queueing mechanism that's important in this case. In the worst-case-scenario, if the child process runs super-fast relative to the GUI, the "result" variable in the child gets overwritten many times, but the parent doesn't care - "result"'s value (ie the address to malloc'ed memory) is simply queued up by PostCallback() for later consumption.

Thank you for taking the time out of your busy schedule to explain and to read this. I really learned a great deal!

Regards,
Ken

Subject: Re: Question about PostCallback from Child Thread
Posted by mirek on Fri, 13 Jul 2007 15:31:28 GMT
View Forum Message <> Reply to Message

Well, two more hints:

- you can also consider whether you should process these data to something else before posting to avoid breaking "get" and "release" library calls into two thread. If not, I think you whould have a lock for the whole library used, you can never say what is going on inside....

- also, this does not sound like a compelling case for multithreading....

Subject: Re: Question about PostCallback from Child Thread
Posted by kfeng on Fri, 13 Jul 2007 16:10:12 GMT
View Forum Message <> Reply to Message

luzr wrote on Fri, 13 July 2007 17:31Well, two more hints:

- you can also consider whether you should process these data to something else before posting to avoid breaking "get" and "release" library calls into two thread. If not, I think you whould have a lock for the whole library used, you can never say what is going on inside....

- also, this does not sound like a compelling case for multithreading....

According to documentation the server on the other side of the library will just continuing to queue prices until your next read.

Now that I understand MT a little bit more, you got me thinking some more. I am currently using a blocking version of the function call to grab prices, but a non-blocking one is also available.

I used to think blocking is always better, but now, I wonder if it is more efficient to poll every second using the non-blocking call in a non-MT app. I think your supposition is right - that the non-blocking will be fast enough that it shouldn't impact user experience in the GUI. A good example would be an application to view your portfolio's profit & loss - don't care if it's a second

late.

However, if latency is important enough that you don't want to set a timer, like an algorithmic black-box for automated trading, then blocking + MT is probably better. But most likely greater latency will show up in other parts of the system, but since it's cumulative, it could be better to squeeze as much as you can out of it. You've opened my eyes again...

I will try both ways next week in the office...

- Ken

Subject: Re: Question about PostCallback from Child Thread
Posted by mirek on Sun, 15 Jul 2007 09:52:05 GMT
View Forum Message <> Reply to Message

BTW, there is a little dirty secret about current PostCallback implementation.... It gets invoked on global timer event only now (each about 10ms, if there are no other GUI events), so you are not going to save any latency compared to simply setting a 10ms timer

---