Subject: Multi-Thread Critial Section Problem
Posted by r1kon on Fri, 09 Dec 2011 21:34:11 GMT
View Forum Message <> Reply to Message

Hello all!

I'm relatively new to C++, so I'm sure there's a much better way of doing this. Also this may be a bit lengthy, but ANY help you can give me would be WONDERFUL as I'm quite stuck on this, and it's something that is being distributed so I need a fix asap.

In my program, I have multiple threads going simultaneously (about 5). It deals with sockets as well. I needed to pass data between the threads as it came in on the sockets..so I did it this way:

Globally (at the top of the cpp file):

```
char recvBuff[100][3][10000];
String arrayOne[100][100][5];
String connBuff[100][7];
String dataInfo[100][10];
String openInfo[100][15];
```

...

Now, when things connect, I assign them a sequence number (I use the socket #) and is stuffed in recvBuff[numConnection][0] as a string (itoa) to be checked on later so it knows what data receiving belongs to which socket. recvBuff handles ALL of the data coming in (which is one of the threads), and will save all of the data in the array into recvBuff[numConnection][1]. The data is checked to make sure it doesn't cross the buffers boundaries.

The second thread is the "workhorse" thread, and will loop through recvBuff to check for new incoming data. Once found, it'll run it through my processing function and act according, then set recvBuff[numConnection][1] back to empty ('\0').

Then, I have a number of different threads that take this data, run work on all of them, and read/write to the global variable buffers above simultaneously.

Also, there are array controls in a few different tabs where the data is spit out, and some of the threads read to/write from these array controls pretty heavily as a way to "pass data" between the threads too.

The program runs just fine for quite some time, no matter what kind of a load I put on it. Sometimes it runs all night with heavy loads coming in on the sockets, all is great.

Sometimes however, a problem occurs (could be anywhere from days to minutes after program start). The program will spontaneously crash, and the error code is (paraphrasing) "read error at 0xHEXHEXHEX" and when run in the debugger, will not jump me to a code location or give me any more information.

Now, I had this problem earlier in the program because one of the threads was reading to/writing from both the global buffers and the array controls quite heavily under some situations, while another thread was simultaneously doing the same. So I consolidated them into the same thread and the problem stopped.

Now, the exact same problem comes up and I can't figure out where.

So now that I've written a novel -- here's my question. Could it be because I'm reading to an array control cell at the same time another thread is writing to it? Or possibly because the buffers are being read to/written from simultaneously by the program?

I've tried to make a program with 2 threads that HEAVILY (in a loop) attempt to read/write to the same location in memory in a global buffer and I haven't been able to replicate the results..so I assume it has to do with the array control?

I'm very much at a loss here, and ANY help at all would be greatly appreciated!

-Kevin

Subject: Re: Multi-Thread Critial Section Problem
Posted by Didier on Sat, 10 Dec 2011 11:34:36 GMT
View Forum Message <> Reply to Message

Hi r1kon,

I suppose you are using mutexes to protect you're shared data ?

If not, then you have to !

Subject: Re: Multi-Thread Critial Section Problem
Posted by koldo on Sat, 10 Dec 2011 12:21:05 GMT
View Forum Message <> Reply to Message

Didier wrote on Sat, 10 December 2011 12:34Hi r1kon,

I suppose you are using mutexes to protect you're shared data ?

If not, then you have to !Oh yes!

You know that when doing multithreading you have to be very rigorous. If not it is like buying tons of lottery... you probably will win

Could you send us a simple testcase and/or samples of how do you do it?

..actually...

I didn't know about mutexes when I made this post. But afterwards I did an hour or so of browsing and I think I got it figured out

The code example..is kind of irrelevant at this point. I know I'm doing it horribly wrong. We're talking stuff like...

Thread #1:
```
 myBuff[i][0] = "STRING";
```

Thread #2:
```
 myBuff[i][0] = "STRING TWO";
```

I'm just straight up reading it, willy nilly.

So here's what I found to do...created some test programs, I think I got it right...

Create some global mutex's (Mutex m1, m2, m3, etc)..I should need one for each buffer and (correct me if I'm wrong here) one for each array ctrl that I'm reading to/writing from (as to avoid reading/writing to the array ctrl at the same time..unless GuiLock __; does this?)

Then, in my code, change (as an example):

```
 myBuff[i][0] = "STRING";
```

to something more like this:

```
void myProgram::setMyBuff(String val)
{
  INTERLOCKED_(m1)
  {
    myBuff[i][0] = val;
  }
}
```

..

```
setMyBuff(t_("STRING"));
```

and similarly:

```
String myProgram::readMyBuff(int pos1, int pos2)
{
  String retVal;
```

```
  INTERLOCKED_(m1)
  {
    retVal = myBuff[pos1][pos2];
  }
  return(retVal);
}

...

String currentValue = readMyBuff(i,0);
```

And if I'm correct..then I need to do these two "functions" (read/write) for each of my threads..and use them EXCLUSIVELY to read/write that way they are ALWAYS syncronized no matter what. Is this correct? And if so, can I use different mutex's (m1, m2, m3, etc) for different buffers?

One more thing -- is it necessary for me to do this with array controls as well? Like so:

```
void myProgram::writeArrayValue(int pos1, int pos2, String val)
{
  INTERLOCK_(m2)
  {
    tabMainTab.arr.Set(pos1,pos2,val);
  }
}
```

So I'm serializing my access to the controls? Or is it just necessary to do it with the data?

Thanks so much in advance! YOu guys, and the other posts on this forum, have been SO helpful!

-Kevin