Subject: ArrayCtrl Write To Freed Memory Detected
Posted by r1kon on Tue, 13 Dec 2011 20:25:46 GMT
View Forum Message <> Reply to Message

I have a program with a few threads, with one in particular that receives data from a socket and writes it to an array control in a tab:

```
splits = Split(work, ',' ,true); //split it all by comma
int theCount = splits.GetCount(); //number

for(int i = 0; i < tabConnectionLayout.arr.GetCount(); i++)
{
 tabConnectionLayout.arr.Set(i,0,splits[1]);
 tabConnectionLayout.arr.Set(i,1,splits[2]);
 tabConnectionLayout.arr.Set(i,2,splits[3]);
 tabConnectionLayout.arr.Set(i,3,splits[4]);
 tabConnectionLayout.arr.Set(i,4,splits[5]); //<-- THIS is where it crashes
}
```

The interesting thing is that the program can run for hours without problems or simply minutes. The crash itself is a PANIC: Writes to freed blocks detected, the break is at the commented line above (I've verified the data is correct, proper sizes in "splits" as well). The log file shows writing to a section of memory that says "FreeFreeFreeFreeFree", etc.

What could be the issue here? I didn't know there would be a problem with writing to an array control like that? Could that be what is happening?

I'm running up the wall on this one, very hard to debug considering the crash is kind of random, but is always the same message. Any help at all would be GREATLY appreciated!

-Kevin

Subject: Re: ArrayCtrl Write To Freed Memory Detected
Posted by dolik.rce on Tue, 13 Dec 2011 21:06:43 GMT
View Forum Message <> Reply to Message

Hi Kevin,

I believe that the piece you're missing is GuiLock. The access to GUI must be serialized, when using multiple threads. See the GuiLock reference example. Second possible solution is to use only one thread to access GUI and send all the actions from threads through a queue. This is illustrated in GuiMT reference example. The GuiLock method is simpler and preferred.

Best regards,
Honza

That's kind of what I figured..I've got GuiLock __; at the top of the workhorse thread that receives all of the data and does MOST of the setting. However, there are still other threads that deal with it.

Here's something that's been bothering me though -- do I need to set GuiLock __; on those that are writing to the array control ONLY?? Or do I need to do it for those that are READING from the array control as well?

The other thing is I set up GuiLock __; on every write to the GUI, and it made my program verrrrryyyy slow to respond (but never crashed! lol). I think it may be due to the fact that I have some indefinite loops and the GuiLock's may be waiting for one another to complete, thus slowing everything down quite a bit.

So, I took them all out but that one (in the workhorse thread)..so it sounds like I need to put them all back, but in a "cleaner" manor.

My question now is..what is the best way to do that? Is it ONLY set to lock within the confines of the nest? For example:

```
for(int i = 0; i < 1000; i++)
{

  for(int z = 0; z < 1000; z++)
  {
    Sleep(1);
    if(something > something_else)
    {
      GuiLock __;
      myArr.Set(z,0,t_("Data to be written"));
    }
    something++;
  }
}
```

Does GuiLock in this case set and then released at the end of the if() statement? Or does it stay set through the entire for() loop? So, the same code snippit..which place would be the best to put GuiLock __;?

```
//HERE? GuiLock __;
for(int i = 0; i < 1000; i++)
{
  //HERE? GuiLock __;
  for(int z = 0; z < 1000; z++)
  {
    Sleep(1);
```

```
    if(something > something_else)
    {
      //Or leave it here? GuiLock __;
      myArr.Set(z,0,t_("Data to be written"));
    }
    something++;
  }
}
```

I hope I'm not being annoying in asking -- I just need to make my multi-thread application as streamlined as possible without it killing the GUI responce.

Thanks in advance..I really appreciate it!

-Kevin

Subject: Re: ArrayCtrl Write To Freed Memory Detected
Posted by dolik.rce on Wed, 14 Dec 2011 06:30:18 GMT
View Forum Message <> Reply to Message

r1kon wrote on Tue, 13 December 2011 22:29That's kind of what I figured..I've got GuiLock __; at the top of the workhorse thread that receives all of the data and does MOST of the setting. However, there are still other threads that deal with it.

Here's something that's been bothering me though -- do I need to set GuiLock __; on those that are writing to the array control ONLY?? Or do I need to do it for those that are READING from the array control as well?
It should be locked for both read and write operations. All of them. Otherwise you could try to read a value while it is being written, which might result in some undefined state.

r1kon wrote on Tue, 13 December 2011 22:29The other thing is I set up GuiLock __; on every write to the GUI, and it made my program verrrrryyyy slow to respond (but never crashed! lol). I think it may be due to the fact that I have some indefinite loops and the GuiLock's may be waiting for one another to complete, thus slowing everything down quite a bit.Yes, that is exactly how it works Whenever one thread is accessing the GUI the rest of the locked areas must wait. There is few tricks how to speed things up. You can limit the scope of the lock, so that it only locks the GUI access, but leaves all the computing heavy parts parallel:myFunc(){

```
    SomeComputing();
    {
        StillParallel();
        GuiLock __;       // GuiLock constructor locks gui access here
        UpdateGui();
    }                     // GuiLock destructor releases the lock automatically at the end of scope
    SomeMoreComputing();
```

}
Also, you might get somewhat better results by speeding up the gui access and of course by limiting the number of writes/reads. This usually leads to some kind of caching, where you store all the results in local variable and output it just a few times per second, all at once. The human user is usually not able to notice more than 10 updates per second anyway

r1kon wrote on Tue, 13 December 2011 22:29So, I took them all out but that one (in the workhorse thread)..so it sounds like I need to put them all back, but in a "cleaner" manor.

My question now is..what is the best way to do that? Is it ONLY set to lock within the confines of the nest? For example:

```
for(int i = 0; i < 1000; i++)
{

  for(int z = 0; z < 1000; z++)
  {
    Sleep(1);
    if(something > something_else)
    {
      GuiLock __;
      myArr.Set(z,0,t_("Data to be written"));
    }
    something++;
  }
}
```

Does GuiLock in this case set and then released at the end of the if() statement? Or does it stay set through the entire for() loop? So, the same code snippit..which place would be the best to put GuiLock __;?

```
//HERE? GuiLock __;
for(int i = 0; i < 1000; i++)
{
  //HERE? GuiLock __;
  for(int z = 0; z < 1000; z++)
  {
    Sleep(1);
    if(something > something_else)
    {
      //Or leave it here? GuiLock __;
      myArr.Set(z,0,t_("Data to be written"));
    }
    something++;
  }
}
```

Yes, as I said above You wan't to limit the scope of the lock to the smallest possible scope. Even by using extra {} where necessary. Also, anything before the GuiLock is constructed is not

affected.

r1kon wrote on Tue, 13 December 2011 22:29I hope I'm not being annoying in asking -- I just need to make my multi-thread application as streamlined as possible without it killing the GUI responce.

Thanks in advance..I really appreciate it!No problem, you are not annoying... and we like to help

Honza
