Subject: MT/Locking Questions
Posted by captainc on Mon, 28 Apr 2008 13:23:02 GMT
View Forum Message <> Reply to Message

This is a general question about using Mutex locking. Does locking occur on a per object basis? what about static variables?

Also, how do locks work in class hierarchies?
For example:

```
class Parent{
  Mutex  lock;
  String mydata;
  virtual void DoSomething(){
     for(int i=0;i<10;++i){
       Cout() << mydata << "\n;"
     }
  };
};

class Child1{
  void DoSomething(){
    INTERLOCKED_(lock){
       mydata = "Foo";
       Parent::DoSomething();
    }
  }
};

class Child2{
 void DoSomething(){
    INTERLOCKED_(lock){
       mydata = "Bar";
       Parent::DoSomething();
    }
  }
};

CONSOLE_APP_MAIN{
  Parent * c1 = new Child1();
  Parent * c2 = new Child2();

  Thread().Run(callback(&c1, &Parent::DoSomething);
  Thread().Run(callback(&c2, &Parent::DoSomething);

  delete c1;
  delete c2;
}
```

Since 2 separate instances of the object are created, both DoSomething()'s can run concurrently,

correct?

Subject: Re: MT/Locking Questions
Posted by mirek on Mon, 28 Apr 2008 17:41:59 GMT
View Forum Message <> Reply to Message

captainc wrote on Mon, 28 April 2008 09:23This is a general question about using Mutex locking. Does locking occur on a per object basis? what about static variables?

Also, how do locks work in class hierarchies?
For example:

```
class Parent{
  Mutex  lock;
  String mydata;
  virtual void DoSomething(){
     for(int i=0;i<10;++i){
       Cout() << mydata << "\n;"
     }
  };
};

class Child1{
  void DoSomething(){
    INTERLOCKED_(lock){
       mydata = "Foo";
       Parent::DoSomething();
    }
  }
};

class Child2{
 void DoSomething(){
    INTERLOCKED_(lock){
       mydata = "Bar";
       Parent::DoSomething();
    }
  }
};

CONSOLE_APP_MAIN{
  Parent * c1 = new Child1();
  Parent * c2 = new Child2();

  Thread().Run(callback(&c1, &Parent::DoSomething);
  Thread().Run(callback(&c2, &Parent::DoSomething);
```

```
  delete c1;
  delete c2;
}
```
Since 2 separate instances of the object are created, both DoSomething()'s can run concurrently, correct?

Yes and you do not even need the lock

General practice for C++ programming is, well, to put it simple:

Ignore MT in the class design as long as you do not need it.

In practice, this means that you need to serialize all write method calls for exclusive access and all read methods calls for shared access if you access single instance from more than one thread concurently.... (client code is responsible for locking).

Mirek

Subject: Re: MT/Locking Questions
Posted by captainc on Mon, 28 Apr 2008 18:00:45 GMT
View Forum Message <> Reply to Message

Yes, that example doesn't need a lock, but I kept it really simple. My larger example that I am working with has objects being created and passed using Ptr<>. A number of threads can act on those objects, and I wanted to be sure that only single object instances are being locked.

Subject: Re: MT/Locking Questions
Posted by mr_ped on Mon, 28 Apr 2008 18:57:24 GMT
View Forum Message <> Reply to Message

For static variables you need a static lock probably?
So it will be shared between instances of the class.

Subject: Re: MT/Locking Questions
Posted by captainc on Mon, 28 Apr 2008 20:10:42 GMT
View Forum Message <> Reply to Message

mr_ped wrote on Mon, 28 April 2008 14:57For static variables you need a static lock probably?
So it will be shared between instances of the class.

You mean declare the lock static?
Ie. "static Mutex lock;"

Subject: Re: MT/Locking Questions
Posted by mr_ped on Tue, 29 Apr 2008 08:03:38 GMT
View Forum Message <> Reply to Message

Well, I never did MT with U++, so I have no direct experience, but that's what makes *sense* to me.

Thinking more about it, the instantiated non-static mutex may be enough, if the multiple threads are working with the same instance of the class with the accessed static variable, but that imposes additional burden on mind of programmer, to never introduce another instance.

Anyway, a search trough files in uppsrc leads to these interesting lines:
C:\upp\uppsrc\Core\heap.cpp(115):static StaticMutex sHeapLock;
C:\upp\uppsrc\Core\Mt.cpp(9): static Mutex *section;
C:\upp\uppsrc\Draw\Draw.cpp(9):static StaticMutex sDrawLock;

As you can see, there's some StaticMutex class also. I'm looking at the source right now, but I have still no idea why ordinary Mutex would be not good enough even for static variable of class.

Sorry.

Subject: Re: MT/Locking Questions
Posted by mirek on Tue, 29 Apr 2008 17:09:55 GMT
View Forum Message <> Reply to Message

mr_ped wrote on Tue, 29 April 2008 04:03
As you can see, there's some StaticMutex class also. I'm looking at the source right now, but I have still no idea why ordinary Mutex would be not good enough even for static variable of class.

Sorry.

The idea begind StaticMutex is this: Normal Mutex has intialization code in constructor. That might cause problems if you are using it as global variable - some global initialization (e.g. INITBLOCK) might try to lock uninitialized mutex.

StaticMutex is a simple wrapper that has no constructor and constructs itself at first call to Enter (in fact, it constructs a regular Mutex inside . However, it can only be used as static or global variable (because it employs initial zero initialization).

Mirek

Subject: Re: MT/Locking Questions
Posted by mr_ped on Tue, 29 Apr 2008 19:49:55 GMT

I bet this info is nowhere in Docs, isn't it?
(didn't find it trough Search on this site: "Mutex -forum")

Edit:
So if I understand you correctly, as long as I don't USE the Mutex in init block, I can freely use "static Mutex some_lock;", right?
If I am sure it will be initialized before I will try to use it, i.e. I will use it only outside of ctors after the application is started, that is.

Subject: Re: MT/Locking Questions
Posted by mirek on Wed, 30 Apr 2008 08:51:50 GMT

mr_ped wrote on Tue, 29 April 2008 15:49I bet this info is nowhere in Docs, isn't it?
(didn't find it trough Search on this site: "Mutex -forum")

Edit:
So if I understand you correctly, as long as I don't USE the Mutex in init block, I can freely use "static Mutex some_lock;", right?
If I am sure it will be initialized before I will try to use it, i.e. I will use it only outside of ctors after the application is started, that is.

Unfortunately, not.

At least, you cannot use it in function body. The problem is that static initialization itself is NOT MT safe.

So e.g.

```
void Fn()
{
   static Mutex x;
}
```

would make Fn require external locking, because two threads might race when checking that flag used to implement static initialization.

Mirek

P.S.: Oh I guess you have in fact covered this issue in your post....

Subject: Re: MT/Locking Questions
Posted by mr_ped on Wed, 30 Apr 2008 14:04:01 GMT

I had only the case

```
class X {
 static Mutex lock_for_A;
 static Value A;
 void function_to_process_A() {
  // ... using that Mutex *HERE*
 }
};
```

on mind

Didn't even think about your static Mutex inside some function, thanks for pointing that out, I could have eventually confused somebody (including myself).

Subject: Re: MT/Locking Questions
Posted by mirek on Wed, 30 Apr 2008 19:05:58 GMT

Yes, this is OK.

Mirek