Subject: Re: Sharing and Locking
Posted by mirek on Tue, 16 Mar 2010 05:02:39 GMT
View Forum Message <> Reply to Message

gridem wrote on Mon, 15 March 2010 16:26
Suppose that we want to share some data between 2 threads. The first thread (SetterThread) will create the global variable and put the pointer to such data, than the data will be destoyed.

I would stop right there and asked "why?" I would prefer using the data instead of pointer.

However, IF I would insist on using the pointer, then the pointer itself is shared resource and you need a lock while accessing it. No matter if it is raw pointer or Pte/Pte.

However, I agree that the existence of weak_ptr::lock is an advantage in some cases, but

boost manual
Even if p.reset() is executed in another thread, the object will stay alive until r goes out of scope or is reset. By obtaining a shared_ptr to the object, we have effectively locked it against destruction.

scares my insticts to the death - this is exactly the case I was speaking about - you are still accessing zombie object that is not supposed to exist anymore.

Quote:
The second (AccesserThread) will try to access to the data and if such data will exist than it will assign some value. From U++ it looks like this:

```
void SetterThread()
{
   while (true)
   {
      Data d;
      *DataAccess() = &d;
   }
}

void AccesserThread()
{
   while (true)
   {
      Ptr<Data> d = *DataAccess();
      if (d)
         d->a = 2;
   }
}
```

Well, obviously, the code is missing serialization of DataAccess...

Now, perhaps we should try hard to add some sort of "Lock" to Ptr and make it wholy atomic, if that is possible. But I do not think that the impact in real world apps would be worth of it.

Mirek
