Subject: Re: Thoughts about alternative approach to multithreading
Posted by Mindtraveller on Wed, 19 Nov 2008 12:20:37 GMT
View Forum Message <> Reply to Message

luzr wrote on Wed, 19 November 2008 12:39
```
StaticMutex              mutex;
VectorMap<String, String> data;

void SetData(const String& key, const String& value)
{
   INTERLOCKED(mutex)
         data.GetAdd(key) = value;
}

String GetData(const String& key)
{
   INTERLOCKED(mutex)
         return data.Get(key, Null);
}
```
What I see here is simply an atomic access to data container.
And it reveals big difference between these two approaches. It is like C and C++. C++ is based on "fully autonomous" objects with a limited number of public methods. Public methods are like high-level messages to the object which is intellectual enough to process them. To keep maintainability, everything should be hidden inside an object. There is still (limited) number of cases where object`s public method is just getting or setting variable. Contrary, Getter/Setter functions commonly are an evil, because usually it equals to making class member public.
...of course, you know it well.

Alternative approach introduces CallbackThread flavour for classes. It adds "multithreading" capability. You still use objects`s public methods, which are now called slightly differently and are processed in the background. From OO perspective you keep classes maintainable since neither public member variables nor Getters/Setters are introduced.
This is contrary to classic MT which commonly uses "shared" variables paired with synchronization objects.
So you can`t compare plain Getter/Setter because well designed objects mustn`t have ones (excluding a number of rare cases described above). You should compare real-world examples, where you don`t just set or get variable, but make some processing with it. In a real-world application you don`t just add to object`s container. Instead you add something to container and do some useful work with it. We have to consider alternative approach is not about handling variables, but doing things.