Subject: Re: Thread calls GUI
Posted by Mindtraveller on Sun, 15 Feb 2009 22:22:57 GMT
View Forum Message <> Reply to Message

Some time ago I`ve started developing "alternative" multithreading system for U++. General idea is that you do not need any sync objects. Because threads do not see ANY shared variables. Instead threads have internal callback queues and exchange with callbacks. Realization is rather optimal (but could be better if I had more spare time).

Recently I was badly needed this approach to be working for a number of threads and also for a "main" GUI thread. Finally these classes are ready and tested for some time, but still under heavy development.

Simplified code looks like this...

1. Declaring main GUI thread/window and one more thread:

```
class GUIThread : public CallbackQueue, public WithMainWindowLayout<TopWindow>
{
public:
 GUIThread();
   ~GUIThread();
 virtual void Init();
 virtual void Shutdown();

 void HandleIncomingMessages();

public /*sync*/:
 void RefreshAll(const Drawing &bdr, const ControlGUI &cg);
 void SetupBathsSettings(Vector<Vector<Value> > rows);
};

class IOThread : public CallbackThread, protected RS232
{
public:
 IOThread();
   ~IOThread();
 virtual void Init();
 virtual void Shutdown();

public /*sync*/:
 void GetAOpState(byte addr);
};
```

Main app function is simple:

```
GUIThread     guiThread;
IOThread      ioThread;
ControlThread controlThread;
```

```
GUI_APP_MAIN
{
 try
 {
  CallbackQueue::InitAll();
  CallbackQueue::StartAll();

  guiThread.Sizeable().Run();

  CallbackQueue::ShutdownAll();
 }
 catch (const Exc &ex)
 {
  PromptOK(ex);
 }
}
```

Finally, if I want any of my threads (including main/GUI) to do something, I just request for this:

```
// i/o therads checks AOp devices and tells their availability to Control thread
void IOThread::GetAOpState(byte addr)
{
 for (int i=0; i<attempts; ++i)
 {
  protoSend[3] = addr;
  protoSend[4] = CMD_AOP_STATUS;
  protoSend.Send(*this, timeout);

  if (protoRecv.Receive(*this, timeout))
  {
   if ((int)protoRecv[3] != (int)addr)
    continue;
   controlThread.Request(&ControlThread::AOpStatus, addr, protoRecv[4]);
  }
 }

 controlThread.Request(&ControlThread::AOpUnavailable, addr);
}

// Control thread analyzes system state and updates GUI accordingly
void ControlThread::SetupDisplayDrawing(bool enableSettings)
{
 static ControlGUI cg;
 // setting controls to be enabled/disabled
 // ...


 // drawing system elements and parameters
```

```
 DrawingDraw d(DISPLAY_W,DISPLAY_H);
 // ...

 guiThread.Request(&GUIThread::RefreshAll, static_cast<Drawing>(d), cg);
}
```

...and no sync objects with their debug.
If this is handy for you, I`ll upload these "alternative" multithreading sources here.
