Subject: Need a suggestion about mouse processing inside threads
Posted by mdelfede on Mon, 12 Dec 2011 23:53:47 GMT
View Forum Message <> Reply to Message

Hi, I've a multi-document application, on which each document runs a threaded command loop :

```
// document command loop
void UppCadDocument::commandLoop(void)
{
 // loop not ended
 INTERLOCKED_(mutex)
 {
  loopEnded = false;
 }
 while(!Thread::IsShutdownThreads())
 {

  if(commandLine.HasCommand())
  {
   String cmd = commandLine.GetCommand();
   if(cmd != "<ESC>" && cmd != "")
    SendCommand(cmd);
  }
  Sleep(100);
 }
 // loop ended
 INTERLOCKED_(mutex)
 {
  loopEnded = true;
 }
}
```

Here, handling of commandLine (some sort of InputField) is simple; the problem arise when I've to react to mouse events; main thread can call, for example, MouseMove() on any time in the middle of anything of my document's loop.

How to synchronize it ? I can't stop my main thread waiting for a document's one is ready for event, of course... I think I'd need some sort of event-loop inside document's threads, and main thread should inject its events on it, instead executing calls directly, but how ?
This is exactly the opposite problem as the one solvable with Guilock....

My idea would be something like this :

Main thread :

void SomeEventHandler(somedata)

```
{
   GetActiveDocument().InjectEvent(eventtype, somedata);
}
```

Document thread :

```
while(true)
{
   WaitForInjectedEventsWithoutEatingCpu();
   while(eventsInQueue)
      ProcessPendingEvents()
}
```

Is there something in upp that can help for all that, or I should code it using arrays of callbacks or something like that ?
Or there's a better solution ?

Max

Subject: Re: Need a suggestion about mouse processing inside threads
Posted by mirek on Tue, 13 Dec 2011 16:47:56 GMT
View Forum Message <> Reply to Message

From what I see, the only thing you need is to synchronize is your commandline. So just create Mutex for reading/writing to commandline...

Not that you will have to make a copy when reading it...

```
void UppCadDocument::commandLoop(void)
{
 // loop not ended
 INTERLOCKED_(mutex)
 {
  loopEnded = false;
 }
 while(!Thread::IsShutdownThreads())
 {
          String currentCommand;
          {
             Mutex::Lock __(commandLineMutex)
        currentCommand = commandLine;
          }
  if(currentCommand.HasCommand())
```

```
  {
   String cmd = currentCommand.GetCommand();
   if(cmd != "<ESC>" && cmd != "")
    SendCommand(cmd);
  }
  Sleep(100);
 }
 // loop ended
 INTERLOCKED_(mutex)
 {
  loopEnded = true;
 }
}
```

Subject: Re: Need a suggestion about mouse processing inside threads
Posted by mdelfede on Tue, 13 Dec 2011 16:53:47 GMT
View Forum Message <> Reply to Message

I guess it's not so simple.... What about mouse events ?
They come from main thread, go into view (which belongs to document running its own thread) and calls document functions asynchronously.

I say that because I tested it and it behaves quite weird.

By now I'm trying to re-direct all gui events from main thread into a FIFO inside each document, which is then processed synchronously by document thread.... Still don't know if it will work as I expect. I see the matter quite tricky....

Max

Subject: Re: Need a suggestion about mouse processing inside threads
Posted by mdelfede on Tue, 13 Dec 2011 17:00:18 GMT
View Forum Message <> Reply to Message

Just to explain a bit more my thoughts....

Managing code (class ThreadQueue, which is a direct parent of UppCadDocument:)

```
// wait for next event and process it
// returns false if shutting down
bool ThreadQueue::WaitAndProcessEvent(void)
{
```

```
 // test again if shutting down
 if(Thread::IsShutdownThreads() || exiting)
  return false;

 // wait for events
 semaphore.Wait();

 // test again if shutting down
 if(Thread::IsShutdownThreads() || exiting)
  return false;

 Callback c;

 // pops next event
 INTERLOCKED_(mutex) {
  ASSERT(!queue.IsEmpty());
  c = queue.Head();
  queue.DropHead();
 }

 // runs the callback
 c.Execute();

 // test again if shutting down
 if(Thread::IsShutdownThreads() || exiting)
  return false;

 return true;
}

// sends an evento to this thread
void ThreadQueue::SendEvent(Callback c)
{
 INTERLOCKED_(mutex) {
  queue.AddTail(c);
  semaphore.Release();
 }
}
```

Document loop :

```
// document command loop
void UppCadDocument::commandLoop(void)
{
 while(WaitAndProcessEvent())
```

```
 {
  if(commandLine.HasCommand())
  {
   String cmd = commandLine.GetCommand();
   if(cmd != "<ESC>" && cmd != "")
    SendCommand(cmd);
  }
 }
}
```

And, for example, a view mouse event handling :

```
////////////////////////////////////////////////////////////////////////////////////////
// middle up - resets pan/3dorbit behaviour
void UppCadView::MiddleUp(Point p, dword keyflags)
{
 SendViewEvent2(uppCadDocument. MiddleUp0, p, keyFlags);
}

void UppCadView::MiddleUp0(Point p, dword keyflags)
{
 isPanning = false;
 PanStartPoint.SetNull();

 isRotating = false;

} // END UppCadView::MiddleUp()
```

(SendViewEvent2 is a macro calling uppCadDocument::SendEvent() with correct parameters)

Subject: Re: Need a suggestion about mouse processing inside threads
Posted by mdelfede on Tue, 13 Dec 2011 17:49:17 GMT
View Forum Message <> Reply to Message

Well... it doesn't work either
Gui is not updated if running inside doc thread....

Subject: Re: Need a suggestion about mouse processing inside threads
Posted by mdelfede on Mon, 19 Dec 2011 14:58:00 GMT
View Forum Message <> Reply to Message

Solved with a separated processong queue per thread

For whom is interested, here a small class :

ThreadQueue.h :

```
#ifndef _ThreadQueue_ThreadQueue_h
#define _ThreadQueue_ThreadQueue_h

#include <Core/Core.h>

using namespace Upp;

class ThreadQueue
{
 private:

  // thread running loop;
  Thread loopThread;

  // loop call function
  void callLoop(Callback cb);

  // the queue/fifo
  BiVector<Callback> queue;

  // controlling semaphore
  Semaphore semaphore;

  // sync mutex
  Mutex mutex;

  // exiting flag -- to manually shutdonwn
  bool exiting;

  // exited flag
  bool exited;

 protected:

 public:

  typedef ThreadQueue CLASSNAME;

  // constructor
  ThreadQueue();

  // destructor
```

```
  ~ThreadQueue();

  // sends an evento to this thread
  void SendEvent(Callback c);

  // wait for next event and process it
  // returns false if shutting down
  bool WaitAndProcessEvent(void);

  // shutdown this thread
  // returns true if successfully shut down after some
  // wait time, false if loop is blocked somehow
  bool Shutdown(void);

  // ask if we're exiting from thread loop
  bool Exiting(void);

  // ask if thread exited
  bool Exited(void);

  // start the command loop
  void StartLoop(Callback cb);
};

#endif
```

ThreadQueue.cpp :

```
#include "ThreadQueue.h"

// constructor
ThreadQueue::ThreadQueue()
{
 // not exited on startup
 exited = false;

 // end not exiting too....
 exiting = false;
}

// destructor
ThreadQueue::~ThreadQueue()
{
 // try to exit loop on destruction, if not already out
 Shutdown();
}
```

```
// loop call function
void ThreadQueue::callLoop(Callback cb)
{
 INTERLOCKED_(mutex) {
  exited = false;
 }
 cb();
 INTERLOCKED_(mutex) {
  exited = true;
 }
}

// wait for next event and process it
// returns false if shutting down
bool ThreadQueue::WaitAndProcessEvent(void)
{
 // test again if shutting down
 if(exiting)
  return false;

 // wait for events
 semaphore.Wait();

 // test again if shutting down
 if(exiting)
  return false;

 Callback c;

 // pops next event
 INTERLOCKED_(mutex) {
  ASSERT(!queue.IsEmpty());
  c = queue.Head();
  queue.DropHead();
 }

 // runs the callback
 c.Execute();

 // test again if shutting down
 if(exiting)
  return false;

 return true;
}

// sends an evento to this thread
```

```
void ThreadQueue::SendEvent(Callback c)
{
 INTERLOCKED_(mutex) {
  queue.AddTail(c);
  semaphore.Release();
 }
}

// shutdown this thread
// returns true if successfully shut down after some
// wait time, false if loop is blocked somehow
bool ThreadQueue::Shutdown(void)
{
 // if already out of loop, just return true
 if(Exited())
  return true;

 // signals that we wanna exit
 INTERLOCKED_(mutex) {
  exiting = true;
 }

 // send a fake event to unlock event waiting
 SendEvent(Callback());

 // wait to give time for loop exiting
 // here about 1s wait max
 for(int i = 0; i < 10; i++)
 {
  if(Exited())
   return true;
  // send a fake event to unlock event waiting
  SendEvent(Callback());
  Sleep(100);
 }

 // if here, thread loop is still busy
 return false;
}

// ask if thread exited
bool ThreadQueue::Exited(void)
{
 bool ex;
 INTERLOCKED_(mutex) {
  ex = exited;
 }
 return ex;
```

```
}

// ask if thread exited
bool ThreadQueue::Exiting(void)
{
 bool ex;
 INTERLOCKED_(mutex) {
  ex = exiting;
 }
 return ex;
}

// start the command loop
void ThreadQueue::StartLoop(Callback cb)
{
 exiting = exited = false;
 loopThread.Start(THISBACK1(callLoop, cb));
}
```

Usage is self explained by comments... I hope.
In short, it's enough to derive a class from ThreadQueue, and you'll have the ability to run a loop which can wait for external messages and process them; the messages are read synchronously one after other like a normal gui message loop.

Here an example ot thread loop function :

```
// document command loop
void UppCadDocument::commandLoop(void)
{
 while(WaitAndProcessEvent())
 {
  if(commandLine.HasCommand())
  {
   String cmd = commandLine.GetCommand();
   if(cmd != "<ESC>" && cmd != "")
    SendCommand(cmd);
   GetCurrentView()->RefreshView();
  }
 }
}
```

The loop waits from an event injected from gui, when it comes check some command line stuff and, if any, process the command inside thread; then it'll stop waiting again.
The WaitAndProcessEvent() function has internal logic to handle a shutdown flag.

Max