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
