Subject: Re: Threading example for U++
Posted by arturbac on Thu, 09 Aug 2007 19:59:36 GMT
View Forum Message <> Reply to Message

I will put here my fresh one hoever this [C++] implementation is not tested, but Same Generic in C# works well.
[Im at porting stage of some app to c++ so why it is stil not tested]

```
template <typename T>
class ThreadSafeQueue
{
 class QueueElement;
public :

 int       GetCount() const;
 void      Enqueue(T value);
 T       Dequeue();
 void      FinishWaiting(bool end_job);

 ThreadSafeQueue();
 ~ThreadSafeQueue();

private:
 int       m_count;
 QueueElement *    m_head;
 QueueElement *    m_tail;
 Threading::Monitor   m_monitor;
 Threading::CriticalSection m_lock;
 bool      m_finish_wait;


 class QueueElement
 {
 private:
  friend class    ThreadSafeQueue;

  T       m_value;
  QueueElement *    m_next;

  T       GetValue() const;

  QueueElement(const T & value);
 };
};
```

```
template <typename T>
int ThreadSafeQueue<T>::GetCount() const
{
 return this->m_count;
}

template <typename T>
ThreadSafeQueue<T>::ThreadSafeQueue()
 : m_head(NULL)
 , m_finish_wait(false)
 , m_tail(NULL)
 , m_count(0)
{}
template <typename T>
ThreadSafeQueue<T>::~ThreadSafeQueue()
{
 m_lock.Enter();
 //Clenup remaining items
 if (this->m_head != NULL)
 {
  QueueElement *p = this->m_head, *pdel;
  while (p != NULL)
  {
   pdel = p;
   p = p->m_next;
   delete pdel;
  }
  this->m_head = NULL;
  this->m_tail = NULL;
 }
 m_lock.Leave();
}
template <typename T>
void ThreadSafeQueue<T>::Enqueue(T value)
{
 m_lock.Enter();
 {
  m_count++;
  QueueElement *enqueue = new QueueElement(value);
  if (m_head == NULL)
  {
   m_head = enqueue;
   m_monitor.Pulse();
  }
  else
   m_tail->m_next = enqueue;
  m_tail = enqueue;
```

```
 }
 m_lock.Leave();
}
template <typename T>
T ThreadSafeQueue<T>::Dequeue()
{
 QueueElement *dequeued;
 m_lock.Enter();
 {
  while (m_head == NULL && !m_finish_wait)
   m_monitor.Wait();
  if (m_head == NULL && m_finish_wait)
  {
   m_lock.Leave();
   return T(); //End
  }
  m_count--;
  dequeued = m_head;
  m_head = m_head->m_next;
 }
 m_lock.Leave();
 T res(dequeued->GetValue());
 delete dequeued;
 return res;
}
template <typename T>
void ThreadSafeQueue<T>::FinishWaiting(bool end_job)
{
 m_lock.Enter();
 {
  m_finish_wait = end_job;
  m_monitor.Pulse();
 }
 m_lock.Leave();
}

template <typename T>
ThreadSafeQueue<T>::QueueElement::QueueElement(const T & value)
 : m_next(NULL)
 , m_value(value)
{}

template <typename T>
T ThreadSafeQueue<T>::QueueElement::GetValue() const
{
 return this->m_value;
}
```

```
class Monitor : public CriticalSection
{
public:

 Monitor();

 void      Wait() throw(...);
 void      Pulse() throw(...);

private:
 int       m_QueueSize;
 CriticalSection    m_lock;
 Semaphore     m_waiting;
};

Monitor::Monitor() : m_QueueSize(0) {}
void Monitor::Wait()
{
 m_lock.Enter();
 m_QueueSize++;
 m_lock.Leave();
 m_waiting.Wait();
}
void Monitor::Pulse()
{
 m_lock.Enter();
 if (m_QueueSize > 0)
  m_waiting.Release();
 m_lock.Leave();
}
```