Producer consumer pattern when many products

Asked 16 years, 1 month ago Modified 16 years, 1 month ago Viewed 1k times

2

I have a producer-consumer pattern working for one product. What is the best implementation when the producer produce many products? For example a DataBaseEvent, GuiEvent and ControlEvent that the consumer shall consume. The code below shows the pattern for one product (a DataBaseEvent). Should each event type be enqueued on an own queue or should the events inherit a base class that can be enqueued. Maybe there exist a better pattern when working with many event types?

```
class DataBaseEventArgs : EventArgs
{
    public string textToDB = "";
}

class Consumer
{
    private Producer mProducer = new Producer();
    private Queue<DataBaseEventArgs> mDataBaseEventQueue = new
Queue<DataBaseEventArgs>();
    private static EventWaitHandle mDataBaseEventWaitHandle = new
EventWaitHandle(false, EventResetMode.ManualReset);
    private Thread mDataBaseEventDequeueThread = null;

    public Consumer()
    {
        mDataBaseEventDequeueThread = new Thread(DataBaseDequeueEvent);
        mDataBaseEventDequeueThread.Start();
        mProducer.mDataBaseEventHandler += WhenDataBaseEvent;
    }

    protected void DataBaseDequeueEvent()
    {
        while (true)
        {
            DataBaseEventArgs e;
            lock (((ICollection)mDataBaseEventQueue).SyncRoot)
            {
                if (mDataBaseEventQueue.Count > 0)
                {
                    e = mDataBaseEventQueue.Dequeue();
                }
            }
            // WriteToDatabase(e.textToDB);
            if (mDataBaseEventQueue.Count == 0)
            {
                mDataBaseEventWaitHandle.WaitOne(1000);
                mDataBaseEventWaitHandle.Reset();
            }
        }
    }
```

```
    internal void WhenDataBaseEvent(object sender, DataBaseEventArgs e)
    {
        lock (((ICollection)mDataBaseEventQueue).SyncRoot)
        {
            mDataBaseEventQueue.Enqueue(e);
            mDataBaseEventWaitHandle.Set();
        }
    }
}

class Producer
{
    public event EventHandler<DataBaseEventArgs> mDataBaseEventHandler = null;

    public void SendDataBaseEvent()
    {
        if (mDataBaseEventHandler != null)
        {
            DataBaseEventArgs e = new DataBaseEventArgs();
            e.textToDB = "This text will be written to DB";
            mDataBaseEventHandler(this, e);
        }
    }
}
```

c# design-patterns events .net-3.5

Share Improve this question Follow

asked Nov 17, 2008 at 9:35

humcfc

349 ● 2 ● 4 ● 11

2 Answers

Sorted by: Highest score (default)

4

Multiple queues would be useful if you want to actively separate the work - i.e. have different threads/pools for different types of event. If you want to share the load, there is another option - use an interface (rather than a base-class). Base-class is fine, but I can't think of anything that would mandate a base-class over an interface. Or even just a delegate to the work to do!

Also - I'm not sure you need a reset event in this case; you can often handle producer/consumer with just lock and Monitor.Pulse / Wait (which has less overhead, as no OS-objects are involved - just managed objects). However, if the code is currently stable, perhaps leave "as is" - threading is hard enough to get right once, let alone twice...

But for reference, it would be something like:

```
while(true) {
    T item;
    lock(lockObj) {
        if(queue.Count == 0) { // empty
            Monitor.Wait(lockObj);
            continue; // ensure there is genuinely something to do
        }
        item = queue.Dequeue();
    }
    // TODO: process item
}
...
void Add(T item) {
    lock(lockObj) {
        queue.Enqueue(item);
        if(queue.Count == 1) { // first
            Monitor.PulseAll(lockObj);
        }
    }
}
```

(and remember to PulseAll when clearing down the queues)

Share

Improve this answer

Follow

edited Nov 17, 2008 at 13:57

answered Nov 17, 2008 at 9:42

Marc Gravell

1.1m ● 273 ● 2.6k ● 3k

Thanks for monitor code sample. I think that I will use that instead of the EventWaitHandler. – humcfc Nov 17, 2008 at 13:39

0

I think that it would be better enqueue all of the request to one queue if they have the same priority and will be handled in similar way.

If one of the requests has higher probability, then either you need some fancy priority queue or you can use different queues.

Also, if you process the messages in a different way, then there is no point to overcomplicate the pattern.

Share Improve this answer Follow

answered Nov 17, 2008 at 9:41

Grzenio

36.6k ● 49 ● 162 ● 241