Application Design Patterns: Producer/Consumer

Publish Date: Sep 19, 2012

Overview

The Producer/Consumer design pattern is based on the Master/Slave pattern, and is geared towards enhanced data sharing between multiple loops running at different rates. As with the standard Master/Slave design pattern, the Producer/Consumer pattern is used to decouple processes that produce and consume data at different rates. The Producer/Consumer pattern's parallel loops are broken down into two categories; those that produce data, and those that consume the data produced. Data queues (which are covered more in-depth in the Queued Message Handler section) are used to communicate data between loops in the Producer/Consumer design pattern. These queues offer the advantage of data buffering between producer and consumer loops.

The Producer/Consumer pattern is commonly used when acquiring multiple sets of data to be processed in order. Suppose you want to write an application that accepts data while processing them in the order they were received. Because queuing up (producing) this data is much faster than the actual processing (consuming), the Producer/Consumer design pattern is best suited for this application. We could conceivably put both the producer and consumer in the same loop for this application, but the processing queue will not be able to add any additional data until the first piece of data is done processing. The Producer/Consumer pattern approach to this application would be to queue the data in the producer loop, and have the actual processing done in the consumer loop. This in effect will allow the consumer loop to process the data at its own pace, while allowing the producer loop to queue additional data at the same time.

This design pattern can also be used effectively when analyzing network communication.
This type of application would require two processes to operate at the same time and at different speeds. The first process would constantly poll the network line and retrieve packets. The second process would take these packets retrieved by the first process and analyze them. In this example, the first process will act as the producer because it is supplying data to the second process which will act as the consumer. This application would benefit from the use of the Producer/Consumer design pattern. The parallel producer and consumer loops will handle the retrieval and analysis of data off the network, and the queued communication between the two will allow buffering of the network packets retrieved. This buffering will become very important when network communication gets busy. With buffering, packets can be retrieved and communicated faster than they can be analyzed.

Table of Contents

1. Why use Producer/Consumer?

The Producer/Consumer pattern gives you the ability to easily handle multiple processes at the same time while iterating at individual rates. What makes this pattern unique is its added benefit of buffered communication between application processes. When there are multiple processes running at different speeds, buffered communication between processes is extremely effective. For example, an application has two processes. The first process performs data acquisition and the second process takes that data and places it on a network. The first process operates at three times the speed as the second process. If the Producer/Consumer design pattern is used to implement this application, the data acquisition process will act as the producer and the network process the consumer. With a large enough communication queue (buffer), the network process will have access to a large amount of the data that the data acquisition loop acquires. This ability to buffer data will minimize data loss.

2. Build a Producer/Consumer

As with the standard Master/Slave pattern, the Producer/Consumer design consists of parallel loops which are broken down into two categories; producers, and consumers. Communication between producer and consumer loops is done by using data queues. LabVIEW has built in queue functionality in the form of VIs in the function palette. These VIs can be found in the function palette under Advance >> Synchronization >> Queue Operations. Queues are based on the first-in/first-out theory. In the Producer/Consumer design pattern, queues can be initialized outside both the producer and consumer loops. Because the producer loop produces data for the consumer loop, it will be adding data to the queue (adding data to a queue is called "enqueue"). The consumer loop will be removing data from that queue (removing data from a queue is called "dequeue"). Because queues are first-in/first-out, the data will always be analyzed by the consumer in the same order as they were placed into the queue by the producer. Figure 1 illustrates how the Producer/Consumer design pattern can be created in LabVIEW.

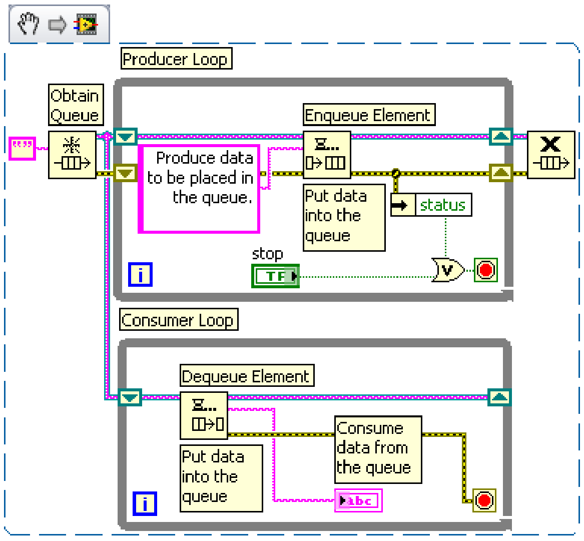

Figure 1: Producer/Consumer Design Pattern

3. Example - Move-Window.vi

This application has the following requirements:

- Create a user interface with four directional control buttons and a queue status indicator.
- Create one loop that collects user interface events and updates the queue indicator. Create another loop that takes the user interface data and moves the window accordingly.

Our first step will be to decide which loop will be the process and which the consumer. Because the user interface is collecting instructional data for another process to carry out, the user interface loop will be the producer. The loop that moves the window depending on user instruction will be the consumer. The producer loop will use a queue to buffer user interface data for the consumer loop. Our application should also monitor instructions that are placed into and removed from the queue.

We are now ready to begin our LabVIEW Producer/Consumer application. To view the final Producer/Consumer application, please open the attached VI (Move-Window.vi).

4. Important Notes

There are some caveats to be aware of when dealing with the Producer/Consumer design pattern such as queue use and synchronization.

- **Queue use**

Problem: Queues are bound to one particular data type. Therefore every different data item that is produced in a producer loop needs to be placed into different queues. This could be a problem because of the complication added to the block diagram.
Solution: Queues can accept data types such as array and cluster. Each data item can placed inside a cluster. This will mask a variety of data types behind the cluster data type. Figure 1 implements cluster data types with the communication queue.

- **Synchronization**

Problem: Since the Producer/Consumer design pattern is not based on synchronization, the initial execution of the loops does not follow a particular order. Therefore, initializing one loop before the other may cause a problem.
Solution: Adding an event structure to the Producer/Consumer design pattern can solve these types of synchronization problems. Figure 2 depicts a template for achieving this functionality. More information pertaining to synchronization functions is located below in the Related Links section.

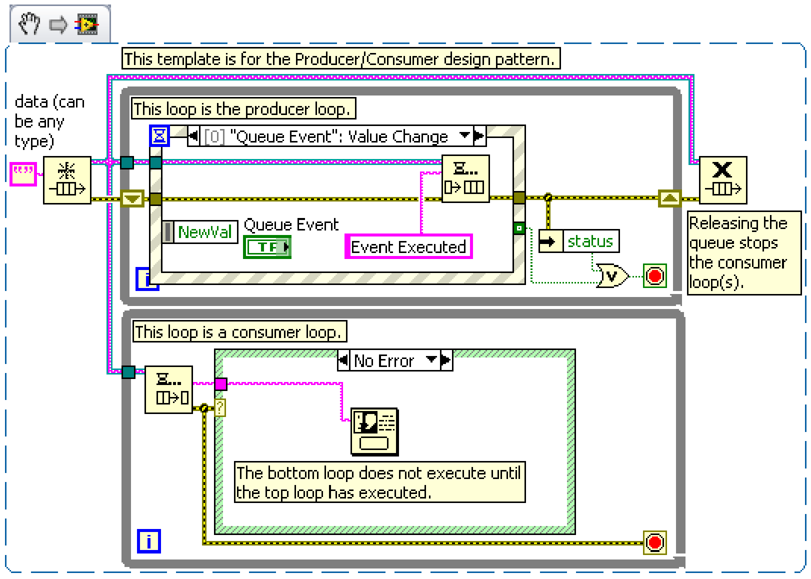
This template is for the Producer/Consumer design pattern.
data (can be any type)
This loop is the producer loop.
[0] "Queue Event": Value Change
NewVal
Queue Event
Event Executed
status
Releasing the queue stops the consumer loop(s).
This loop is a consumer loop.
No Error
The bottom loop does not execute until the top loop has executed.

Figure 2: Using an Event Structure in Producer/Consumer Design Pattern

Related Links:
Technical Presentation: LabVIEW Application Design Patterns
LabVIEW Discussion Forum
LabVIEW 2011 Help: Synchronization VIs and Functions