# The posted message queue vs the input queue vs the message queue

devblogs.microsoft.com/oldnewthing/20130530-00

May 30, 2013

Raymond Chen

There are multiple ways of viewing the interaction between posted messages and input messages. MSDN prefers to view posted messages and input messages as part of one giant pool of messages in a *message queue*, with rules about which ones get processed first. I, on the other hand, prefer to think of posted messages and input messages as residing in different queues that are processed in sequence.

By analogy, consider a business with a policy that loyalty program members are served ahead of regular customers.

One way of organizing this is to form a single queue, but sorting them so members go to the front. To call the next person in line, you just take whoever is at the head of the queue.

```
AddToQueue(Customer c)
{
 if (c is a member and the queue contains non-members) {
  let n = the first non-member in the queue;
  insert c in front of n;
 } else {
  insert c at the end of the queue;
 }
}


GetNextCustomer()
{
 if (there is somebody in the queue) {
  return the first person in the queue;
 }
 // nobody is waiting
 return null;
}
```

This approach works fine from a programmatic standpoint, but people might not like it when they see others cutting in front of them. You might therefore choose to create two queues, one for members and one for non-members:

```
AddToQueue(Customer c)
{
 if (c is a member) {
  insert c at end of member queue;
 } else {
  insert c at end of non-member queue;
}


GetNextCustomerInLine()
{
 if (there is somebody in the member queue) {
  return the first person in the member queue;
 }
 if (there is somebody in the non-member queue) {
  return the first person in the non-member queue;
 }
 // nobody is waiting
 return null;
}
```

Note that this second algorithm serves customers in exactly the same order; the only difference is psychological. Customers might resent the presence of a *members* line since it reminds them that other people are getting special treatment. Or this algorithm may not be practical because you don't have room in your lobby for two lines. You might choose yet another algorithm, where you select members at the time the customers are served rather than when they arrive. Everybody forms a single queue, but sometimes a customer in the middle of the queue gets pulled out for special treatment. (Maybe they wear red hats so they are easy to spot.)

```
AddToQueue(Customer c)
{
 insert c at end of queue;
}


GetNextCustomerInLine()
{
 if (there is a member in the queue) {
  return the first member in the queue;
 }
 // else only non-members remaining
 if (there is somebody in the queue) {
  return the first person in the queue;
 }
 // nobody is waiting
 return null;
}
```

(Some banks use a variant of this algorithm to separate potential bank-robbers from the rest of the customers.)

From the standpoint of determining who gets served in what order, all of these algorithms are equivalent. One may be more efficient, one may be easier to understand, one may be easier to maintain, one may be easier to debug, but they all accomplish the same thing.

So go ahead and use whatever interpretation of the message queue you like. They are all equivalent.

But the interpretation that I use is the one I presented several years ago at the PDC, where the posted message queue and input queues are considered separate queues. I prefer that interpretation because it makes input queue attachment easier to understand.

In the analogy above, I prefer viewing things in terms of the second algorithm, with separate queues; MSDN prefers viewing things in terms of the third algorithm, with a single queue, but with some messages given preferential treatment over others.

But as I've already noted, the outputs of the algorithms are the same given the same inputs, so it doesn't matter how they are implemented internally, since all you can observe as a program are the inputs and outputs. It's like the multiple interpretations of quantum mechanics: They all attempt to describe the world from a particular viewpoint, but at the end of the day, the world simply *is*, and a description is just a description.

Next time, we'll explore some consequences of the interaction between posted messages and input messages.

Raymond Chen

## Follow