

Re: Thread Checking the Queue data in an infinite loop

Source: <http://www.tech-archive.net/Archive/VC/microsoft.public.vc.mfc/2008-01/msg00594.html>

- *From:* "Doug Harrison [MVP]" <dsh@xxxxxxxx>
 - *Date:* Sat, 12 Jan 2008 12:56:50 -0600
-

On Sat, 12 Jan 2008 10:00:38 -0800 (PST), prams <pramodjoisb@xxxxxxxx> wrote:

Others who commented on My solution using Event Object:

Yes I want to drain the Queue once I complete the wait..Since the whole idea is whenever there is data in the Queue, We have to retrieve and Process it. So the signalling of event Objects by the Writer threads will create unnecessary wakeups and I have to check whether the queue is empty or not. I will do the check whether the queue is empty or not.

Remember also that the reader has to acquire the mutex.

My colleague in another project who is implementing something similar to my project has used Pipes which signal an event whenever there is data. When he had told me about this during the beginning of project, I had told him that I want to use some simple things Like queue not wanting to use IPC Mechanisms between threads. Now I think he was right in choosing that.

Using pipes will require you to serialize the data to transport it over a byte stream. On the other hand, pipes will impose a bound on your buffer, which I don't believe you mentioned in your original post. Absent an explicit size limit, the only thing preventing your queue from growing out of control is the amount of actual data, and if that is unbounded, the behavior and fairness of the Windows scheduler. If you want to do some more research, google "bounded buffer"; that's the name of the problem you're trying to solve. Here's a link:

http://en.wikipedia.org/wiki/Producer-consumer_problem

Note that they are using the term "mutex" as a synonym for "binary semaphore"; while common in textbooks, this is incorrect for Windows,

Re: Thread Checking the Queue data in an infinite loop

because a mutex is owned by the thread that locks it and can only be unlocked by that thread, while semaphores do not have this thread affinity. So you'd have to use actual semaphores to implement their second algorithm.

Note also that when you ask to read N bytes from a pipe, and the pipe contains data, you may retrieve any number from between 1 and N bytes, even before EOF is reached. So if you expect to read, say, a 32 byte data structure, you better do it in a loop, where each iteration decreases the requested amount by the number of bytes read by previous iterations and moves the buffer pointer by the same amount.

Now with very few days left (1 week) , I have to find a proper solution which is not very Complicated and will solve my problem without us having to retest a lot of things which we have already done with our code.

1) Is Sleep(1) solution not ok at all?

It could be described as "quick and dirty".

2) Use of Event Synchronization is not perfect. Should i not go ahead with it?

As long as you deal with the things I mentioned, it'll work. You actually described sort of an imperfect simulation of condition variables, which unfortunately don't exist natively in Windows before Vista. (The Wiki article's "monitor" example can be interpreted in terms of condition variables. Implicit in a "monitor" class is the locking of a mutex by every function, so if you were to ignore the "monitor" attribute and lock a mutex in each function, you would have the condition variable solution.)

3) I should use a Pipe or some other mechanism like I/O Completion port (PostQueuedCompletionStatus/GetQCS)

I would not do that.

—

Doug Harrison
Visual C++ MVP

.