

Re: Thread deadlock misery

Source: <http://www.tech-archive.net/Archive/VC/microsoft.public.vc.language/2007-03/msg00437.html>

- *From:* "PaulH" <paul.heil@xxxxxxxxx>
 - *Date:* 13 Mar 2007 08:20:06 -0700
-

I used the critical section because I read in another post on this group that it might be necessary despite the Interlocked() operation being atomic. In this case it obviously doesn't help me any, so I'll remove it. Thanks!

The Sleep() call in the transmit thread is to regulate the number of transmissions per second. The Sleep() call in the StatCalc thread is to regulate the number of statistical samples per second.

Below is some actual code from my transmit thread. After it sends about 300 frames, this thread halts.

```
/// count of frames sent
volatile long m_nSentFrames;

////////////////////////////////////
// Method public static CMainFrame::TransmitThread
/// @brief Transmit unicast frames to a client at a specified rate.
///
/// @param TFP – [in] allocated pointer to the parameters
/// @return DWORD WINAPI – 0
////////////////////////////////////
/*static*/ DWORD WINAPI CMainFrame::TransmitThread( TTCB* TFP )
{
    std::auto_ptr< TTCB > pATFP( TFP );

    WSADATA wsaData = { 0 };
    WSASStartup( MAKEWORD( 2, 2 ), &wsaData );

    //
    // Get the address info for the DUT
    //
    struct addrinfo *DUTAddr = NULL,
    hints = { 0 };
    hints.ai_family = AF_UNSPEC;
    hints.ai_socktype = SOCK_DGRAM;
    hints.ai_protocol = IPPROTO_UDP;
    char cPort[ 10 ] = { 0 };
    _itoa_s( TFP->port, cPort, 9, 10 );
```

Re: Thread deadlock misery

```
if( getaddrinfo( pATFP->TFP.address, cPort, &hints, &DUTAddr ) !=
0 )
{
// error handling
}

//
// open a UDP, DGRAM socket for the connection
//
SOCKET DUTSocket = socket( AF_INET, SOCK_DGRAM, IPPROTO_UDP );
if( DUTSocket < 0 )
{
// error handling
}

//
// create an empty buffer to pad the frames to the appropriate
size
//
std::vector< char > buffer( pATFP->TFP.size );
WSABUF wsaBuf = { pATFP->TFP.size, &buffer.front() };

//
// Throw unicast frames at the given rate & size to the DUT
//
CMainFrame* pParent = reinterpret_cast< CMainFrame* >( pATFP-

        pParent );

while( pATFP->pParent->m_bRunning )
{
DWORD dwStartTime = GetTickCount();
DWORD sent = 0;
if( WSASendTo( DUTSocket,
&wsaBuf,
1,
&sent,
0,
DUTAddr->ai_addr,
sizeof( SOCKADDR ),
NULL,
NULL ) < 0 )
{
// error handling
}
InterlockedIncrement( &pParent->m_nSentFrames );
DWORD dwSleepTime = ( 1000 / pATFP->TFP.FPS ) -
( GetTickCount() - dwStartTime );
if( dwSleepTime > 0 )
Sleep( dwSleepTime );
}
```

Re: Thread deadlock misery

```
//  
// Test finished successfully, cleanup  
//  
closesocket( DUTSocket );  
freeaddrinfo( DUTAddr );  
WSACleanup();  
return 0;  
}
```

If you can tell where I'm going wrong here, please let me know.

Thanks,

PaulH

On Mar 12, 5:29 pm, "Doug Harrison [MVP]" <d...@xxxxxxx> wrote:

On 12 Mar 2007 15:12:30 -0700, "PaulH" <paul.h...@xxxxxxx> wrote:

I have an application with 4 threads GUI, Transmit, Receive, and StatCalc. The transmit and receive threads both update a counter variable, and the StatCalc thread periodically updates some statistics based on those two counters. Unfortunately, the Transmit and StatCalc threads deadlock and only the receive thread runs.

The GUI thread initiates the Transmit, Receive, and StatCalc threads virtually simultaneously.

Below is some pseudo-code of what I'm doing

```
_Transmit_  
While(TRUE) {  
    send a frame;  
    EnterCriticalSection(&TransmitCounterCrit);  
    InterlockedIncrement(&TransmitCounter);  
    LeaveCriticalSection(&TransmitCounterCrit);  
    Sleep(20);  
}
```

```
_Receive_  
While(TRUE){
```

Re: Thread deadlock misery

```
receive some frames;
EnterCriticalSection(&ReceiverCounterCrit);
InterlockedExchangeAdd(&ReceiveCounter, framecount);
LeaveCriticalSection(&ReceiverCounterCrit);
wait for next frame;
}
```

```
_StatCalc_
While(TRUE) {
//wait for the Transmit and Receive threads to generate data
Sleep(500);
calculate statistics;
```

```
//reset the counters
EnterCriticalSection(&TransmitCounterCrit);
InterlockedExchange(&TransmitCounter, 0);
LeaveCriticalSection(&TransmitCounterCrit);
EnterCriticalSection(&ReceiverCounterCrit);
InterlockedExchange(&ReceiveCounter, 0);
LeaveCriticalSection(&ReceiverCounterCrit);
}
```

If anybody see a way to prevent these threads from dead locking, let me know.

In order for a deadlock to occur, there must be a circular wait. There is none in what you've shown. In addition, you're guarding individual InterlockedXXX function calls with a mutex, and this is unnecessary. The whole point of the InterlockedXXX functions is to obviate mutexes under these conditions. The problem is thus in code you haven't shown, perhaps in the code that receives or transmits. Also, why are you calling Sleep?

--
Doug Harrison
Visual C++ MVP