

Multithreaded pipe server, deadlock problem

Source:

<http://www.tech-archive.net/Archive/Development/microsoft.public.win32.programmer.kernel/2008-04/msg00400.htm>

- *From:* "Bill Holt" <mailbill(NOSPAM)@21cn.com.nospam>
 - *Date:* Tue, 29 Apr 2008 08:24:24 +0800
-

Hi,

I'm working on a multithreaded pipe server. I need help debugging a dead lock problem.

Pipe server's primary thread is as below:

```
#define g_nPipeBufferSize 8192
HANDLE m_hStopEvent; // This is created and maintained elsewhere to allow shutting down of the pipe
server

OVERLAPPED ol = {0};
HANDLE hArray[2];
hArray[0] = ol.hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);
hArray[1] = m_hStopEvent;
BOOL bStop = FALSE;
while (bStop == FALSE)
{
HANDLE hServerPipe = CreateNamedPipe(L"\\\\.\\Pipe\\MyServerPipe",
PIPE_ACCESS_DUPLEX|FILE_FLAG_OVERLAPPED|WRITE_DAC,
PIPE_TYPE_BYTE | PIPE_READMODE_BYTE | PIPE_WAIT, PIPE_UNLIMITED_INSTANCES,
g_nPipeBufferSize, g_nPipeBufferSize, NMPWAIT_USE_DEFAULT_WAIT, NULL);
if (hServerPipe && hServerPipe != INVALID_HANDLE_VALUE)
{
BOOL bClientConnected = ConnectNamedPipe(hServerPipe, &ol);
DWORD dwOverlappedBytes;
switch (WaitForMultipleObjects(2, hArray, FALSE, INFINITE))
{
case WAIT_OBJECT_0:
ResetEvent(hArray[0]);
if (GetOverlappedResult(hServerPipe, &ol, &dwOverlappedBytes, TRUE))
{
DWORD dwThreadId;
HANDLE hThread = CreateThread(NULL, 0, InstanceThread, (LPVOID)hServerPipe, 0, &dwThreadId);
if (hThread == NULL || hThread == INVALID_HANDLE_VALUE)
bStop = TRUE;
else
CloseHandle(hThread);
}
}
}
}
```

Multithreaded pipe server, deadlock problem

```
break;
case WAIT_OBJECT_0+1:
bStop = TRUE;
CloseHandle(hServerPipe);
break;
}
}
}
CloseHandle(hArray[0]);
```

The primary thread create name pipes and if a client is connected, it then create a thread and let the thread procedure "InstanceThread" process the clients. The "InstanceThread" is created using the code below:

```
BOOL bStop = FALSE;
BYTE bBuffer[g_nPipeBufferSize];
DWORD cbBytes;
OVERLAPPED ol = {0};
HANDLE hArray[2];
hArray[0] = ol.hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);
hArray[1] = m_hStopEvent;
while(bStop == FALSE)
{
ZeroMemory(bBuffer, sizeof(bBuffer));
BOOL bResult = ReadFile( hPipe, bBuffer, sizeof(bBuffer), NULL, &ol);
DWORD dwError = GetLastError();
if (!bResult && dwError == ERROR_IO_PENDING)
{
// Marker 0
switch (WaitForMultipleObjects(2, hArray, FALSE, INFINITE))
{
case WAIT_OBJECT_0:
ResetEvent(hArray[0]);
GetOverlappedResult(hPipe, &ol, &cbBytes, TRUE);
if(cbBytes>0)
_ProcessQuery(bBuffer);
break;
case WAIT_OBJECT_0+1:
bStop = TRUE;
break;
}
if(!_IsQueryComplete())
break;
}
else
bStop = TRUE;
}
if (_IsQueryComplete())
{
vector<BYTE> vbOut;
_ProduceOutput(vbOut);
if (vbOut.size()>0)
```

Multithreaded pipe server, deadlock problem

```
{
WriteFile( hPipe, &vbOut[0], vbOut.size(), &cbBytes, &ol);
switch (WaitForMultipleObjects(2, hArray, FALSE, INFINITE))
{
case WAIT_OBJECT_0:
ResetEvent(hArray[0]);
break;
case WAIT_OBJECT_0+1:
break;
}
}

}
CloseHandle(hArray[0]);
FlushFileBuffers(hPipe);
DisconnectNamedPipe(hPipe);
CloseHandle(hPipe);
```

And below is the client that connects to this pipe:

```
HANDLE hPipe = CreateFile( L"\\\\.\\Pipe\\MyServerPipe", GENERIC_READ | GENERIC_WRITE, 0,
NULL, OPEN_EXISTING, FILE_FLAG_OVERLAPPED, NULL);
if(hPipe && hPipe != INVALID_HANDLE_VALUE)
{
DWORD cbBytes;
OVERLAPPED ol = {0};
ol.hEvent = CreateEvent(NULL, TRUE, TRUE, NULL);
BOOL bResult = WriteFile( hPipe, szPipeQuery.c_str(), szPipeQuery.length(), &cbBytes, &ol);
if (!bResult && GetLastError() == ERROR_IO_PENDING)
bResult = GetOverlappedResult(hPipe, &ol, &cbBytes, TRUE);

BYTE bOutBuffer[8192];
while(true)
{
// Marker 1
BOOL bResult = ReadFile( hPipe, bOutBuffer, sizeof(bOutBuffer), &cbBytes, &ol);
// Marker 2
if (!bResult && dwError == ERROR_IO_PENDING)
bResult = GetOverlappedResult(hPipe, &ol, &cbBytes, TRUE);
if (dwLastError == ERROR_BROKEN_PIPE || dwLastError == ERROR_NO_DATA || dwLastError ==
ERROR_PIPE_NOT_CONNECTED)
{
CloseHandle(hPipe);
hPipe = NULL;
break;
}
if (dwLastError == ERROR_INVALID_HANDLE) /* is handle already closed? */
break;
if(cbBytes>0)
_ProcessResponse(bOutBuffer);
if(!_IsResponseValid())
```

Multithreaded pipe server, deadlock problem

```
break;
}
CloseHandle(ol.hEvent);
if(hPipe && hPipe!=INVALID_HANDLE_VALUE)
CloseHandle(hPipe);
}
```

My pipe server works like the HTTP server. It uses a two way pipe. A client sends a query and the server response with result. As you can see in the code that both the server and the client is asynchronous. The reason why I need asynchronous access is that I need the ability to gracefully stop the server thread. Microsoft did provide an example but it's in synchronous mode, making it impossible to quit because ConnectNamedPipe is always blocking. And please note that after a successfully processed query, the server will always close down the pipe connection.

The server and client does work. But the problem appears in stress test. When we have two or three clients accessing the server pipe at the same time, queries are processed almost instantly. The client will encounter a dead lock at certain point. I ran this test on a single computer.

The deadlock is around Marker 1 and Marker 2 in the client code. I traced down, and found that when the deadlock is because of GetOverlappedResult never returns in Marker 2. This means the pipe server never sends back the response. However, further trace showed that although "CreateFile" and "WriteFile" calls in the client returned successful results, the server never received the query. At the deadlock, the server thread stops at Marker 0, which means it's still waiting for the client to send queries. It's hard to believe that the server didn't receive the query. But it looks like it's the cause of the deadlock. Is there a hole in my code that can cause this problem? How do I debug this issue? Should I check some where else?

I used Windbg to make sure the deadlock is not cause by critical sections. My environment is Vista x64 with Visual Studio 2008. Haven't tested on other platforms.

Thank you very much for looking into my problem.

Best regards,
Bill Holt

.