Note: The Client demo below is currently nonfunctional. The code was accidentally truncated, and I am going to have to re-write it, due to a crash on my computer. I will also rewrite the server to add clarity to the way it works. Apologies for any inconvenience.

-

thedarkfreak Jan 23, 2010

This is a demo of using Named Pipes for interprocess communication in LB. It is translated almost directly from the Microsoft VB.NET example: http://support.microsoft.com/kb/871044

This has two programs, obviously to demonstrate the way it works. The first one I called PipeServer.bas, but the name doesn't really matter :P

```
openMode = _PIPE_ACCESS_DUPLEX or _FILE_FLAG_WRITE_THROUGH
pipeMode = _PIPE_WAIT or _PIPE_TYPE_MESSAGE or _PIPE_READMODE_MESSAGE
hPipe = CreateNamedPipe("MyPipe"
, openMode, pipeMode, 10, 10000, 2000, 10000, 0)
struct a, num as long
BUFFSIZE = 10000
buffer$ = "WHEE"
buffer$ = buffer$ + space$(BUFFSIZE - 4)
print hPipe
If hPipe = 0 then goto [end]
Do
 res = ConnectNamedPipe(hPipe, 0)
 cbnCount = 4
 res$ = ReadFile$(hPipe, len(a.struct), cbnCount)
 byteCount = val(res$)
 print res$
 If byteCount > BUFFSIZE then byteCount = BUFFSIZE
 res = WriteFile(hPipe, buffer$, byteCount)
 res = FlushFileBuffers(hPipe)
 res = DisconnectNamedPipe(hPipe)
Loop Until byteCount = 0
a = CloseHandle(hPipe)
[end]
end

Function CreateNamedPipe(
lpName$, dwOpenMode, dwPipeMode, nMaxInstances, nOutBufferSize, nInBuf
ferSize, nDefaultTimeOut, lpSecurityAttributes)
```

```
 lpName$ = "\\.\pipe\";lpName$
 CallDLL #kernel32, "CreateNamedPipeA",_
 lpName$ as ptr,_
 dwOpenMode as long,_
 dwPipeMode as long,_
 nMaxInstances as long,_
 nOutBufferSize as long,_
 nInBufferSize as long,_
 nDefaultTimeOut as long,_
 lpSecurityAttributes as long,_
 CreateNamedPipe as ulong
End Function

Function ConnectNamedPipe(hNamedPipe, lpOverlapped)
 CallDLL #kernel32, "ConnectNamedPipe",_
 hNamedPipe as ulong,_
 lpOverlapped as long,_
 ConnectNamedPipe as long
End Function

Function DisconnectNamedPipe(hNamedPipe)
 CallDLL #kernel32, "DisconnectNamedPipe",_
 hNamedPipe as ulong,_
 DisconnectNamedPipe as long
End Function

Function WriteFile(hFile, buf$, size)
 struct num, bytesWritten as long
 CallDLL #kernel32, "WriteFile",_
 hFile as ulong,_
 buf$ as ptr,_
 size as long,_
 num as struct,_
 lpOverlapped as long,_
 WriteFile as long
End Function

Function ReadFile$(hFile, size, byref num)
 buf$ = space$(size)
 struct num, bytesRead as long
 CallDLL #kernel32, "ReadFile",_
 hFile as ulong,_
 buf$ as ptr,_
 size as long,_
 num as struct,_
 lpOverlapped as long,_
```

```
 ret as long

 ReadFile$ = trim$(buf$)
 num = num.bytesRead.struct
End Function

Function FlushFileBuffers(hFile)
 CallDLL #kernel32, "FlushFileBuffers",_
 hFile as ulong,_
 FlushFileBuffers as long
End Function

Function CloseHandle(hHandle)
 CallDLL #kernel32, "CloseHandle",_
 hHandle as ulong,_
 CloseHandle as long
End Function
```

Note that this program hangs while listening, the debugger says it hangs at the "ConnectNamedPipe" API call until another process writes something to it.

This one I named PipeClient. Please note that this isn't the snipped posted on the Microsoft site, this is just how I did it based on what they said(they said you can use CreateFile, ReadFile and WriteFile to work with it, then give a different example showing transacted pipes using CallNamedPipe. I do it the first way)

```
'Open the pipe
hPipe = OpenPipe("MyPipe")

'This I was using to make sure it connected right.
Print hPipe

'This tells the "server" we'll have a buffer set up for 256 bytes.
str$ = "256"
size = len(str$)
Print WriteFile(hPipe, str$, size)

'This actually reads the 256 bytes.
'The num is passed byref, and will hold the number of bytes acutally r
ead.
'Because of the way the server is set up, this should print WHEE.
Print ReadFile$(hPipe, 256, num)

'This should print 256 on the screen.
Print num
```

```
'The code on the server is set up so if it recieves a 0 as
'the amount of bytes to send, it closes, so this is how you get the fi
rst one
'out of the infinite loop and terminate.
Print WriteFile(hPipe, "0", 1)

'All handles opened through windows have to be specifically closed.
a = CloseHandle(hPipe)


Function OpenPipe(lpName$)
'I guessed the parameters for this due to the similarity to
'opening logical drives for reading/writing. That's also possible in
'LB, and
```