WinPipes

Introduction

The winpipes DLL is a tool that provides IPC communications to Microsoft Windows 3.0 and is designed to emulate stream file functions under 'C.'

While pipes arean integral part of many operating systems, Windows lacks this facility. Windows relies almost exclusively on DDE and the clipboard for application communication. While these methods are fine for large complicated pieces of data, pipes are simple structures and require very little brain work to incorporate. With a pipe, communication between processes is as simple as file I/O .

Microsoft Windows imposes many restrictions that other operating systems do not. One such restriction is non-preemptive multitasking and another is the inability to "block" or stop a process while waiting. Because of this I had to rethink how pipes should work under Windows. In UNIX, a pipe will block the reading process until data is written to it. Since Windows will not block a process, how do you wait for pipe data? My solution was to send read and write messages. When a window writes to a pipe, a WM_USER message is sent to the read window with the number of bytes, the window that has written them, and the wNotify number used when the pipe was opened or created. When a window reads a pipe, a WM_USER message is sent to the write window with the number of bytes read, the window that has read them, and the wNotify number used when the pipe was opened or created. More about this later.

The winpipes cannot use global memory. Since it is a DLL it cannot own global memory. All pipes combined are limited to less than 64K. If you need large data transfer, use DDE or the clipboard. Networks are also a problem. While real operating systems incorporate pipes as part of the file system, winpipes are not part of Windows'. A winpipe can not communicate via a network to another machine unless **you** want to write a PIPE Server application.

Lastly, winpipes was originally written as a debugging tool. While I am as sure as any Windows developer can be that it is reasonably bug free, there is always a chance Microsoft overlooked a subtlety, or even, God forbid, I could have made a mistake. So please try it out and let me know what you think.

Contact me:
Mark L. Woodward
Compuserve I.D. 72760, 2400

Licensing

You may use this DLL as you wish on site as a debugging or educational tool.

Mark L. Woodward
51 Florida Avenue
Plymouth, MA 02360

If you wish to discuss distribution or licensing the source code, contact me through the previously mentioned Compuserve account.

What is a Winpipe?

A winpipe handle, PIPE, is an index into a static structure array within the DLL's data segment. This structure contains, among other things, the handle to the pipes buffer, the read window and its wNotify value, and the write window with its wNotify value. The pipe itself is a combination of a FIFO buffer and the message management system.

Creating a Pipe

Creating a pipe consists of defining its name, its buffer size, and what you want to do with it.

lpszPipeName is the name, under 16 bytes, by which other applications can access your pipe. This name will be used by other applications when opening the pipe.

wStyle is a WORD combination of pipe styles. The styles can be PIPE_READ, PIPE_WRITE, and PIPE_MONITOR. PIPE_READ creates the pipe with your window as the read window. No other window will be able to read your pipe. PIPE_WRITE creates the pipe with your window as the write window. No other window will be able to write to your pipe. PIPE_MONITOR creates a pipe with no owners. Until someone opens it with either PIPE_READ or PIPE_WRITE, any window will be able to read it or write to it.

wBufferSize is the buffer size of the pipe. Use this sparingly. All pipes combined are limited to 64K, minus some static memory. It is recommended that you use either the default size or just a little more than the largest item you wish to communicate.

wNotify is the wParam sent to the window depending on wStyle. If wNotify is zero, no messages will be sent. If wStyle is PIPE_MONITOR, this is not used.

Pipe Notification

Pipe notification is the process in which windows are notified that a pipe needs attention. When a window has opened a pipe as either PIPE_READ or PIPE_WRITE and specified a wNotify value greater than zero, it will receive messages when the pipe is ready. If a window has read rights to a pipe, a message will be sent to it every time the pipe is written to. The message will be a WM_USER, the wParam will be the wNotify value, and the lParam will contain the number of bytes in the LOWORD and the write window's handle in the HIWORD. If no window has write rights to the pipe, this number is zero.

Closing a Pipe

When a pipe is closed, the window that is closing it is deleted from the pipe's user list. If the pipe has no more users, it will be deleted and its buffer freed. Note: if a pipe is opened as PIPE_MONITOR by one window and opened as PIPE_READ by another, the read window's "ClosePipe" call will delete the pipe. PIPE_MONITORS have no entry into the user list. Unless you want to delete a pipe, use ReleasePipe. ReleasePipe acts the same as ClosePipe except that the pipe is never deleted. Never use ClosePipe on a standard pipe.

Standard Pipes

Standard pipes are simply pipes that you do not have to create. Stdin, Stdout, and Stderr are all pipes created at winpipe's startup. The example source provided, stdio and testapp, use Stdin and Stdout to communicate information to each other. testapp uses these standard pipes in a way that is consistent with the naming convention. Stdio uses them backwards - as a server. Stdio is a text messaging server that can display debugging messages from applications and provide back door debugging dialog to an application during the development stage. Stdio also has a log file option so debugging messages can be saved to stdio.log in your root directory.

Example Programs

Stdio and testapp are the examples given. Although not magazine quality, they are fairly readable and should be commented enough to use as examples of pipe usage.

What they do:

1) Testapp will echo all characters typed in its window to stdio via Stdout.

2) Stdio will send characters typed in its window to testapp via the Stdin pipe, testapp will echo back via Stdout.

3) Testapp has a debugging dialog routine that responds to commands typed in the Stdio window. Some of these are "who:" "what:" "stat:."

4) Stdio can save information passed to it by opening a log file in the File menu.