win32-console-docs Public

master Branches Tags Go to file Add file Code

10 Commits

src

.gitattributes

README.md

README

Console Handles and Standard Handles

This document attempts to explain how console handles and standard handles work, and how they interact with process creation and console attachment and detachment. It is based on experiments that I ran against various versions of Windows from Windows XP to Windows 10.

The information here is verified by the test suite in the `src` directory. It should be taken with a grain of salt. I don't have access to many operating systems. There may be important things I didn't think to test. Some of the behavior is surprising, so it's hard to be sure I have fully identified the behavior.

Feel free to report errors or omissions. An easy thing to do is to run the accompanying test suite and report errors. The test suite is designed to expect bugs on the appropriate Windows releases.

Table of Contents

Common semantics

There are three flags to `CreateProcess` that affect what console a new console process is attached to:

- `CREATE_NEW_CONSOLE`
- `CREATE_NO_WINDOW`
- `DETACHED_PROCESS`

These flags are interpreted to produce what I will call the *CreationConsoleMode*. `CREATE_NO_WINDOW` is ignored if combined with either other flag, and the combination of `CREATE_NEW_CONSOLE` and `DETACHED_PROCESS` is an error:

Criteria	Resulting *CreationConsoleMode*
None of the flags (parent has a console)	*Inherit*
None of the flags (parent has no console)	*NewConsole*
`CREATE_NEW_CONSOLE`	*NewConsole*
`CREATE_NEW_CONSOLE` and `CREATE_NO_WINDOW`	*NewConsole*
`CREATE_NO_WINDOW`	*NewConsoleNoWindow*
`DETACHED_PROCESS`	*Detach*
`DETACHED_PROCESS` and `CREATE_NO_WINDOW`	*Detach*
`CREATE_NEW_CONSOLE` and `DETACHED_PROCESS`	none - the `CreateProcess` call fails
All three flags	none - the `CreateProcess` call fails

Windows' behavior depends on the *CreationConsoleMode*:

- *NewConsole* or *NewConsoleNoWindow*: Windows attaches the new process to a new console. *NewConsoleNoWindow* is special--it creates an invisible console. (Prior to Windows 7, `GetConsoleWindow` returned a handle to an invisible window. Starting with Windows 7, `GetConsoleWindow` returns `NULL`.)
- *Inherit*: The child attaches to its parent's console.
- *Detach*: The child has no attached console, even if its parent had one.

I have not tested whether or how these flags affect non-console programs (i.e. programs whose PE header subsystem is `WINDOWS` rather than `CONSOLE`).

There is one other `CreateProcess` flag that plays an important role in understanding console handles -- `STARTF_USESTDHAND` This flag influences whether the `AllocConsole` and `AttachConsole` APIs change the "standard handles" (`STDIN/STDOUT/STDERR`) during the lifetime of the new process, as well as the new process' initial standard handles, of course. The standard handles are accessed with `GetStdHandle` and `SetStdHandle`, which are effectively wrappers around a global `HANDLE[3]` variable -- these APIs do not use `DuplicateHandle` or `CloseHandle` internally, and while NT kernels objects are reference counted, `HANDLE`s are not.

The `FreeConsole` API detaches a process from its console, but it never alters the standard handles.

(Note that by "standard handles", I am strictly referring to `HANDLE` values and not `int` file descriptors or `FILE*` file streams provided by the C language. C and C++ standard I/O is implemented on top of Windows `HANDLE`s.)

Traditional semantics

Console handles and handle sets (traditional)

In releases prior to Windows 8, console handles are not true NT handles. Instead, the values are always multiples of four minus one (i.e. 0x3, 0x7, 0xb, 0xf, ...), and the functions in `kernel32.dll` detect the special handles and perform LPCs to `csrss.exe` and/or `conhost.exe`.

A new console's initial console handles are always inheritable, but non-inheritable handles can also be created. The inheritability can be changed, except on Windows 7 (see [win7inh]).

About

Win32 Console Documentation -- in particular, console/standard handles and CreateProcess inheritance

Readme
Activity
111 stars
10 watching
16 forks
Report repository

Releases

No releases published

Packages

No packages published

Languages

C++ 94.4% C 3.7% Makefile 1.9%

Traditional console handles cannot be duplicated to other processes. If such a handle is used with `DuplicateHandle`, the source and target process handles must be the `GetCurrentProcess()` pseudo-handle, not a real handle to the current process.

Whenever a process creates a new console (either during startup or when it calls `AllocConsole`), Windows replaces that process' set of open console handles (its *ConsoleHandleSet*) with three inheritable handles (0x3, 0x7, 0xb)
Whenever a process attaches to an existing console (either during startup or when it calls `AttachConsole`), Windows completely replaces that process' *ConsoleHandleSet* with the set of inheritable open handles from the originating process
These "imported" handles are also inheritable.

CreateProcess (traditional)

The manner in which Windows sets standard handles is influenced by two flags:

- Whether `STARTF_USESTDHANDLES` was set in `STARTUPINFO` when the process started (*UseStdHandles*)
- Whether the `CreateProcess` parameter, `bInheritHandles`, was `TRUE` (*InheritHandles*)

From Window XP up until Windows 8, `CreateProcess` sets standard handles using the first matching rule:

1. If *UseStdHandles*, then the child uses the `STARTUPINFO` fields. Windows makes no attempt to validate the handles, nor will it treat a non-inheritable handle as inheritable simply because it is listed in `STARTUPINFO`.
2. If *ConsoleCreationMode* is *NewConsole* or *NewConsoleNoWindow*, then Windows sets the handles to (0x3, 0x7, 0xb).
3. If *ConsoleCreationMode* is *Detach*, then Windows sets the handles to (`NULL`, `NULL`, `NULL`).
4. If *InheritHandles*, then the parent's standard handles are copied as-is to the child, without exception.
5. Windows duplicates each of the parent's non-console standard handles into the child. Any standard handle that looks like a traditional console handle, up to 0x0FFFFFFF, is copied as-is, whether or not the handle is open. [1]

   If Windows fails to duplicate a handle for any reason (e.g. because it is `NULL` or not open), then the child's new handle is `NULL`. The child handles have the same inheritability as the parent handles. These handles are not closed by `FreeConsole`. (Bugs: [xppipe] [xpinh] [dupproc] [wow64dup])

The `bInheritHandles` parameter to `CreateProcess` does not affect whether console handles are inherited. Console handles are inherited if and only if they are marked inheritable. The `PROC_THREAD_ATTRIBUTE_HANDLE_LIST` attribute added in Vista does not restrict console handle inheritance, and erratic behavior may result from specifying a traditional console handle in `PROC_THREAD_ATTRIBUTE_HANDLE_LIST`'s `HANDLE` list. (See the `Test_CreateProcess_InheritList` test in `src`.)

AllocConsole, AttachConsole (traditional)

`AllocConsole` and `AttachConsole` set the standard handles as follows:

- If *UseStdHandles*, then Windows does not modify the standard handles.
- If !*UseStdHandles*, then Windows changes the standard handles to (0x3, 0x7, 0xb), even if those handles are not open.

FreeConsole (traditional)

After calling `FreeConsole`, no console APIs work, and all previous console handles are apparently closed -- even `GetHandleInformation` fails on the handles. `FreeConsole` has no effect on the `STDIN/STDOUT/STDERR` values.

Modern semantics

Console handles (modern)

Starting with Windows 8, console handles are true NT kernel handles that reference NT kernel objects. Console handles are associated with a device path beginning with `\Device\ConDrv\`. View the full path using `procexp.exe` or `handle.exe` from sysinternals (e.g. `handle.exe -a -p <pid>`). Be sure to run the tool elevated, or you will only see the `\Device\ConDrv` for each handle.

If a process is attached to a console, then it will have two open console handles that Windows uses internally -- one to `\Device\ConDrv\Connect` and another to `\Device\ConDrv\Reference`.

Ordinary I/O console objects can be classified in two ways:

- *Input* vs *Output*
- *Bound* vs *Unbound*

A *Bound Input* object is tied to a particular console, and a *Bound Output* object is tied to a particular console screen buffer. These objects are usable only if the process is attached to the correct console. *Bound* objects are created through these methods only:

- `CreateConsoleScreenBuffer` (associated with `\Device\ConDrv\ScreenBuffer`)
- opening `CONIN$` or `CONOUT$` (associated with `\Device\ConDrv\CurrentIn` and `\Device\ConDrv\CurrentOut`)

Most console objects are *Unbound*, which are created during console initialization. For any given console API call, an *Unbound Input* object refers to the currently attached console's input queue, and an *Unbound Output* object refers to the screen buffer that was active during the calling process' console initialization. These objects are usable as long as the calling process has any console attached.

Unbound objects are associated with `\Device\ConDrv\Input` and `\Device\ConDrv\Output`.

Unlike traditional console handles, modern console handles **can** be duplicated to other processes.

CreateProcess (modern)

Whenever a process is attached to a console (during startup, `AttachConsole`, or `AllocConsole`), Windows will sometimes create new *Unbound* console objects and assign them to one or more standard handles. If it assigns to both `STDOUT` and `STDERR`, it reuses the same new *Unbound Output* object for both.

As with previous releases, standard handle determination is affected by the *UseStdHandles* and *InheritHandles* flags.

Each of the child's standard handles is set using the first match:

1. If *InheritHandles*, *UseStdHandles*, and the relevant `STARTUPINFO` field is non-`NULL`, then Windows uses the `STARTUPINFO` field. As with previous releases, Windows makes no effort to validate the handle, nor will it treat a non-inheritable handle as inheritable simply because it is listed in `STARTUPINFO`. [2]
2. If *CreationConsoleMode* is *NewConsole* or *NewConsoleNoWindow*, then Windows opens a handle to a new *Unbound* console object. This handle will be closed if `FreeConsole` is later called. (N.B.: Windows reuses the same *Unbound* output object if it creates handles for both `STDOUT` and `STDERR`. The handles themselves are still different, though.)
3. If *ConsoleCreationMode* is *Detach*, then Windows sets the handle to `NULL`.
4. If *UseStdHandles*, the child's standard handle becomes `NULL`.
5. If *InheritHandles*, and there is no `PROC_THREAD_ATTRIBUTE_HANDLE_LIST` specified, then the parent's standard handle is copied as-is.
6. The parent's standard handle is duplicated. As with previous releases, if the handle cannot be duplicated, then the child's handle becomes `NULL`. The child handle has the same inheritability as the parent handle. `FreeConsole` does *not* close this handle, even if it happens to be a console handle (which is not unlikely). (Bugs: [dupproc])

AllocConsole, AttachConsole (modern)

`AllocConsole` and `AttachConsole` set the standard handles as follows:

- If *UseStdHandles*, then Windows opens a console handle for each standard handle that is currently `NULL`.
- If !*UseStdHandles*, then Windows opens three new console handles.

Implicit screen buffer refcount

When a process' console state is initialized (at startup, `AllocConsole` or `AttachConsole`), Windows increments a refcount on the console's currently active screen buffer, which decrements only when the process detaches from the conso
All *Unbound Output* console objects reference this screen buffer.

FreeConsole (modern)

As in previous Windows releases, `FreeConsole` in Windows 8 does not change the `STDIN/STDOUT/STDERR` values. If Windows opened new console handles for `STDIN/STDOUT/STDERR` when it initialized the process' console state, then `FreeConsole` will close those handles. Otherwise, `FreeConsole` will only close the two internal handles.

Interesting properties

- `FreeConsole` can close a non-console handle. This happens if:
  i. Windows had opened handles during console initialization.
  ii. The program closes its standard handles and opens new non-console handles with the same values.
  iii. The program calls `FreeConsole`.

  (Perhaps programs are not expected to close their standard handles.)
- Console handles--*Bound* or *Unbound*--can be duplicated to other processes. The duplicated handles are sometimes usable, especially if *Unbound*. The same *Unbound Output* object can be open in two different processes and refer to different screen buffers in the same console or in different consoles.
- Even without duplicating console handles, it is possible to have open console handles that are not usable, even with a console attached.
- Dangling *Bound* handles are not allowed, so it is possible to have consoles with no attached processes. The console cannot be directly modified (or attached to), but its visible content can be changed by closing *Bound Output* handles to activate other screen buffers.
- A program that repeatedly reinvoked itself with `CREATE_NEW_CONSOLE` and `bInheritHandles=TRUE` would accumulate console handles. Each child would inherit all of the previous child's console handles, then allocate three more for itself. All of the handles would be usable (if the program kept track of them somehow).

Other notes

SetActiveConsoleScreenBuffer

Screen buffers are referenced counted. Changing the active screen buffer with `SetActiveConsoleScreenBuffer` does not increment a refcount on the buffer. If the active buffer's refcount hits zero, then Windows chooses another buffer and activates it.

`CREATE_NO_WINDOW` process creation flag

The documentation for `CREATE_NO_WINDOW` is confusing:

> The process is a console application that is being run without a console window. Therefore, the console handle for the application is not set.
>
> This flag is ignored if the application is not a console application, or if it is used with either `CREATE_NEW_CONSOLE` or `DETACHED_PROCESS`.

Here's what's evident from examining the OS behavior:

- Specifying both `CREATE_NEW_CONSOLE` and `DETACHED_PROCESS` causes the `CreateProcess` call to fail.
- If `CREATE_NO_WINDOW` is specified together with `CREATE_NEW_CONSOLE` or `DETACHED_PROCESS`, it is quietly ignored, just as documented.
- Otherwise, `CreateProcess` behaves the same way with `CREATE_NO_WINDOW` as it does with `CREATE_NEW_CONSOLE`, except that the new console either has a hidden window (before Windows 7) or has no window at all (Windows 7 and later). These situations can be distinguished using the `GetConsoleWindow` and `IsWindowVisible` calls. `GetConsoleWindow` returns `NULL` starting with Windows 7.

`PROC_THREAD_ATTRIBUTE_HANDLE_LIST`

The `PROC_THREAD_ATTRIBUTE_HANDLE_LIST` list cannot be empty; the `UpdateProcThreadAttribute` call fails if `cbSize` is `0` However, a list containing a `NULL` is apparently OK and equivalent to an empty list. Curiously, if the inherit list has both a non-`NULL` handle and a `NULL` handle, the list is still treated as empty (i.e. the non-`NULL` handle is not inherited).

Starting with Windows 8, `CreateProcess` duplicates the parent's handles into the child when `PROC_THREAD_ATTRIBUTE_HANDLE_LIST` and these other parameters are specified:

- *InheritHandles* is true
- *UseStdHandles* is false
- *CreationConsoleMode* is *Inherit*

Bugs

Windows XP does not duplicate a pipe's read handle [xppipe]

On Windows XP, `CreateProcess` fails to duplicate a handle in this situation:

- `bInheritHandles` is `FALSE`.
- `STARTF_USESTDHANDLES` is not specified in `STARTUPINFO.dwFlags`.
- One of the `STDIN/STDOUT/STDERR` handles is set to the read end of an anonymous pipe.

In this situation, Windows XP will set the child process's standard handle to `NULL`. The write end of the pipe works fine Passing a `bInheritHandles` of `TRUE` (and an inheritable pipe handle) works fine. Using `STARTF_USESTDHANDLES` also works See `Test_CreateProcess_Duplicate_XPPipeBug` in `src/HandleTests` for a test case.

Windows XP duplication inheritability [xpinh]

When `CreateProcess` in XP duplicates an inheritable handle, the duplicated handle is non-inheritable. In Vista and later, the new handle is also inheritable.

`CreateProcess` duplicates `INVALID_HANDLE_VALUE` until Windows 8.1 [dupproc]

From Windows XP to Windows 8, when `CreateProcess` duplicates parent standard handles into the child, it duplicates `INVALID_HANDLE_VALUE` (aka the `GetCurrentProcess()` pseudo-handle) to a true handle to the parent process. This bug was fixed in Windows 8.1.

On some older operating systems, the WOW64 mode also translates `INVALID_HANDLE_VALUE` to `NULL`.

CreateProcess duplication broken w/WOW64 [wow64dup]

On some versions of 64-bit Windows, when a 32-bit program invokes another 32-bit program, `CreateProcess`'s handle duplication does not occur. Traditional console handles are passed through, but other handles are converted to `NULL`. The problem does not occur when 64-bit programs invoke 64-bit programs. (I have not tested 32-bit to 64-bit or vice versa.)

The problem affects at least:

- Windows 7 SP1

Windows Vista BSOD

It is easy to cause a BSOD on Vista and Server 2008 by (1) closing all handles to the last screen buffer, then (2) creating a new screen buffer:

```
#include <windows.h>
int main() {
    FreeConsole();
    AllocConsole();
    CloseHandle((HANDLE)0x7);
    CloseHandle((HANDLE)0xb);
    CreateConsoleScreenBuffer(
        GENERIC_READ | GENERIC_WRITE,
        FILE_SHARE_READ | FILE_SHARE_WRITE,
        NULL,
        CONSOLE_TEXTMODE_BUFFER,
        NULL);
    return 0;
}
```

Windows 7 inheritability [win7inh]

- Calling `DuplicateHandle(bInheritHandle=FALSE)` on an inheritable console handle produces an inheritable handle, but it should be non-inheritable. Previous and later Windows releases work as expected, as does Windows 7 with a non-console handle.
- Calling `SetHandleInformation(dwMask=HANDLE_FLAG_INHERIT)` fails on console handles, so the inheritability of an existing console handle cannot be changed.

Windows 7 `conhost.exe` crash with `CONOUT$` [win7_conout_crash]

There is a bug in Windows 7 involving `CONOUT$` and `CloseHandle` that can easily crash `conhost.exe` and/or activate the wrong screen buffer. The bug is triggered when a process without a handle to the active screen buffer opens `CONOUT$` and then closes it using `CloseHandle`.

Here's what *seems* to be going on:

Each process may have at most one "console object" referencing a particular buffer. A single console object can be shared between multiple processes, and whenever console handles are imported (`CreateProcess` and `AttachConsole`), the objects are reused.

If a process opens `CONOUT$` , however, and does not already have a reference to the active screen buffer, then Windows creates a new console object. The bug in Windows 7 is this: if a process calls `CloseHandle` on the last handle for a console object, then the screen buffer is freed, even if there are other handles/objects still referencing it At that point, the console might display the wrong screen buffer, but using the other handles to the buffer can return garbage and/or crash `conhost.exe` . Closing a dangling handle is especially likely to trigger a crash.

Rather than using `CloseHandle` , letting Windows automatically clean up a console handle via `FreeConsole` or exiting somehow avoids the problem.

The bug affects Windows 7 SP1, but does not affect Windows Server 2008 R2 SP1, the server version of the OS.

See `src/HandleTests/Win7_Conout_Crash.cc` .

Test suite

To run the test suite, install Cygwin or MSYS2, and install the MinGW-w64 G++ compiler package. Enter the `src` subdirect
Run `./configure` , then `make` , and finally `build/HandleTests.exe` .

For a WOW64 run:

- Build the 64-bit `Worker.exe` .
- Rename it to `Worker64.exe` and save it somewhere.
- Build the 32-bit binaries.
- Copy `Worker64.exe` to the `build` directory alongside `Worker.exe` .

Footnotes

1: From the previous discussion, it follows that if a standard handle is a non-inheritable console handle, then the child's standard handle will be invalid:

- Traditional console standard handles are copied as-is to the child.
- The child has the same *ConsoleHandleSet* as the parent, excluding non-inheritable handles.

It's an interesting edge case, though, so I test for it specifically. As of Windows 8, the non-inheritable console handle would be successfully duplicated.

2: Suppose a console program invokes `CreateProcess` with these parameters:

- `bInheritHandles` is `FALSE` .
- `STARTF_USESTDHANDLES` is set.
- `STARTUPINFO` refers to inheritable console handles (e.g. the default standard handles)

Prior to Windows 8, the child would have received valid standard handles. As of Windows 8, the child's standard handles will be `NULL` instead.

Copyright