What is the maximum numeric value for a socket, and what is the maximum number of sockets a Windows program can create?

devblogs.microsoft.com/oldnewthing/20161221-00

December 21, 2016

Raymond Chen

A customer had a problem with their application that used network sockets, and they wanted to know what the maximum numeric value for a socket is. "The program uses a signed integer to hold the socket descriptor, and we found in our testing that the numeric value of `INVALID_SOCKET` is `0xFFFFFFFF`. What is the maximum value?"

In addition to being a vague question, it's also a strange question, so we asked for more information about the problem they are having, in the hopes that we could both understand how the problem led them to asking the strange question, and so we could try to solve the problem.

The customer explained that they have a multithreaded application that uses thousands of network sockets. After running for several days, the customer observed that socket operations are failing with `INVALID_SOCKET`, and `WSAGetLastError` returns error 10038: `WSAENOTSOCK`. Since the error is intermittent, the customer is under the impression that the application may have created so many sockets that their socket numbers have exceeded the maximum legal numeric value for a socket, resulting in the `INVALID_SOCKET` error.

The customer added, "According to this link, the maximum number of sockets that a program can use is determined at compile time by the manifest constant `FD_SETSIZE`. However, we cannot find where this constant is defined."

Okay, it's not clear where the customer is getting the impression that a single program cannot use more than `FD_SETSIZE` sockets. Indeed, the documentation they referenced says quite the opposite:

> The maximum number of sockets that a Windows Sockets application can use is **not** affected by the manifest constant FD_SETSIZE.

(Emphasis mine.)

The documentation continues:

> This value defined in the Winsock2.h header file is used in constructing the **FD_SET** structures used with **select** function. The default value in Winsock2.h is 64.

Which conveniently answers the customer's third question.

What the `FD_SETSIZE` constant determines is the maximum number of sockets that can be passed in a single call to the `select` function. The total number of sockets available to a program is not limited by `FD_SETSIZE`.

And as the documentation notes, you can make `FD_SETSIZE` bigger if you need to. The point is that the `fd_set` structure is a variable-sized structure, but for compatibility with Unix programs, it is formally defined as a fixed-size structure so that programs can pass them around.

Okay, now back to the original question: Is it possible that the `socket` function is returning socket numbers that are not legal, and that's why the program gets `INVALID_SOCKET` when it tries to perform socket operations on those sockets?

This is another case of starting with the assumption that you found an operating system bug instead of starting with the assumption that you have a bug in your program.

While it's possible that there is a bug in the operating system code that does socket management that causes it to hand out invalid socket handles, a much more likely reason that your program is being told that it is using invalid socket handles is, um, because it is using invalid socket handles.

Verify that the handle being passed really is a valid socket. Maybe it was closed prematurely elsewhere. Maybe there is a bug in some other part of the code that is double-closing a handle (and the second time it closes, it accidentally closed your socket handle). Or maybe there is a bug in some other part of the code that is closing an uninitialized handle variable, so it's basically rolling the dice, and most of the time it gets `ERROR_INVALID_HANDLE`, but once in a while, the uninitialized handle variable happens to contain a value that numerically matches one of your socket handles, and it ended up accidentally closing your socket.

If you really believe that the `socket` function is returning invalid sockets, I guess you can add debugging code that takes the return value of every call to `socket` and (if it is not `INVALID_SOCKET` indicating that the system could not create a socket) call `getsockopt` to read an arbitrary-selected socket option, and see whether it fails with `WSAENOTSOCK`.

I bet it won't. The socket handle was probably good at the point the system gave it to you. You probably did something to make it go bad. Application Verifier can help you find out what.

Raymond Chen

Follow