tdistler.com

"To err is human, but to really foul things up you need a computer."

Cross-Platform IPv6 Socket Programming

On February 28, 2011, in Code Monkey, by Tom

Writing code that works on Windows, Linux, and Mac is frequently challenging. Socket programming is no exception. Modern versions of Linux and Mac have full implementations of the latest IPv6 socket API extensions defined in RFC 3493. Windows, however, has only a partial implementation of the original (deprecated) version, RFC 2553. This sounds worse than it is, but it's something you have to consider.

Note: This post assumes you are already familiar with the socket extensions for IPv6 (RFC 3493).

Linux and Mac
Good news… they both fully support RFC 3493.

Windows
Windows IPv6 support varies based on which version you're targeting. Microsoft started adding IPv6 in Windows 2000, and they've continued adding more of the socket extensions as time

went on. Most of the core functionality is present in XP, and what's missing is easily replaced by using Winsock calls directly (more on this later).

Windows gained IPv6 support while RFC 2553 was still the supported standard. Since then, it has been deprecated by RFC 3493. However, Microsoft doesn't want to break existing code written against it's API, so the older API lives on. The main impact of this is that `sockaddr_in6` and `sockaddr_storage` are slightly different on Windows than Mac and Linux. The size of the structures across platforms is the same (the `sa_family_t` member was shortened), it's just that the Windows structures don't begin with the length member. For example:

```
// Linux and Mac
struct sockaddr_in6 {
    uint8_t      sin6_len;    /* Added in RFC 3493 */
    sa_family_t  sin6_family;
    ...
};
struct sockaddr_storage {
    uint8_t      ss_len;      /* Added in RFC 3493 */
    sa_family_t  ss_family;
    ...
};

// Windows
struct sockaddr_in6 {
    sa_family_t  sin6_family;
    ...
};
struct sockaddr_storage {
    sa_family_t  ss_family;
    ...
};
```

I've never had a problem with this, because the size of `sockaddr_in6` is easily determined (`sizeof(sockaddr_in6)`) and I always end up casting `sockaddr_storage` to the specific type (`sockaddr_in` or `sockaddr_in6`) based on `ss_family`.

Besides the data structure differences, it's important to remember that Microsoft added IPv6 support over multiple versions. Support first appeared in Windows 2000, but more of the extensions have been added over time. Most of the core functionality was present in XP (including multicast), but not everything is implemented as of Windows 7. It's annoying, but I will say that what's missing is easily replaced by using Winsock calls directly.

Here's the breakdown of IPv6 socket extensions by Windows version:

Socket Extension	2K	XP	Vista	7	Comments
if_indextoname()			x	x	GetAdaptersAddresses() for XP
if_nametoindex()			x	x	GetAdaptersAddresses() for XP
if_nameindex()					GetAdaptersAddresses() (XP, later)

if_freenameindex()					
getaddrinfo()	x	x	x	x	
getnameinfo()	x	x	x	x	
freeaddrinfo()	x	x	x	x	
gai_strerror()	x	x	x	x	
inet_pton()			x	x	WSAStringToAddress() (2000, XP)
inet_ntop()			x	x	WSAAddressToString() (2000, XP)
All IN6_IS_ADDR_* macros		x	x	x	Ex: IN6_IS_ADDR_LOOPBACK()
struct sockaddr_storage		x	x	x	
Multicast support		x	x	x	

As you can see, you may still have to call Winsock directly depending on what version of Windows you are targeting. In my opinion, programming IPv6 on Windows is a lot easier if you only support XP and later, but I know that’s not always possible.

Summary

Modern operating systems all support IPv6. However, for business reasons, Windows has a slightly older version of the socket API which requires special consideration. My goal was to enumerate those differences to help make the transition to IPv6 smoother. Writing cross-platform code can be a fun challenge at times, but it’s also a little tedious. Hopefully, this post helps ease the pain.