Convert an IP address to a human-readable string in C

Content

Tested on

Debian (Lenny)
Ubuntu (Precise, Trusty)

Objective

To convert an IPv4 or IPv6 address to a human-readable string (for example `192.168.0.1` or `2001:db8::1`)

Scenario

Suppose you have used the `getpeername` function to obtain the remote address to which a particular TCP socket is connected:

```
struct sockaddr_storage addr;
socklen_t addr_len=sizeof(addr);
int err=getpeername(sock_fd,(struct sockaddr*)&addr,&addr_len);
if (err!=0) {
    die("failed to fetch remote address (errno=%d)",errno);
}
```

The remote address has been written to a buffer called `addr`. This buffer is of type `struct sockaddr_storage`, but the address stored within it will be of type `struct sockaddr_in` or `sockaddr_in6`. The length of the address has been recorded in the variable `addr_len`. Note that:

- `addr` is a socket address, so in addition to the IP address it contains information such as the address family and port number.
- `addr_len` will probably not be equal to `sizeof(struct sockaddr_storage)` once the call to `getpeername` has completed.

You wish to convert the IP address contained within `addr` to a human-readable string.

Method

One way to perform the required conversion is to call the `getnameinfo` function. By default this attempts to

convert the address into a domain name, however it can be instructed to produce a numeric address instead by setting the `NI_NUMERICHOST` flag:

```
#include <netdb.h>
#include <sys/socket.h>
#include <netinet/in.h>

// ...

char buffer[INET6_ADDRSTRLEN];
int err=getnameinfo((struct sockaddr*)&addr,addr_len,buffer,sizeof(buffer),
    0,0,NI_NUMERICHOST);
if (err!=0) {
    die("failed to convert address to string (code=%d)",err);
}
printf("Remote address: %s\n",buffer);
```

The string buffer needs to be at least `INET_ADDRSTRLEN` bytes long for IPv4 and `INET6_ADDRSTRLEN` for IPv6. Since these constants are fixed (by POSIX) at 16 and 46 bytes respectively, `INET6_ADDRSTRLEN` can be presumed to suffice for either address family.

Variations

Converting IPv4-mapped IPv6 addresses to plain IPv4

If an IPv4 connection is made to an IPv6 socket then the local and remote network addresses will be represented as IPv4-mapped addresses. For example, the IPv4 address `192.168.0.1` would be represented by the IPv6 address `::ffff:192.168.0.1`.

This format is readable, but it is probably not the best choice for presentation to the user. Since the connection was made using IPv4, the user could reasonably expect to see an IPv4 address. This can be achieved by converting the address from IPv6 to IPv4 before calling `getnameinfo`:

```
if (addr.ss_family==AF_INET6) {
    struct sockaddr_in6* addr6=(struct sockaddr_in6*)&addr;
    if (IN6_IS_ADDR_V4MAPPED(&addr6->sin6_addr)) {
        struct sockaddr_in addr4;
        memset(&addr4,0,sizeof(addr4));
        addr4.sin_family=AF_INET;
        addr4.sin_port=addr6->sin6_port;
        memcpy(&addr4.sin_addr.s_addr,addr6->sin6_addr.s6_addr+12,sizeof(addr4.sin_addr.s_addr));
        memcpy(&addr,&addr4,sizeof(addr4));
        addr_len=sizeof(addr4);
    }
}
```

The conversion is performed only if the address family is IPv6, and then only if the address if IPv4-mapped. The address buffer must be writable, and of the appropriate size and alignment to hold an IPv4 or IPv6 socket address. (That is the case here because the buffer is of type `struct sockaddr_storage`).

Alternatives

Using inet_ntop

An alternative method is to use the function `inet_ntop`. This is somewhat easier to use than `getnameinfo`

if the IP address is not already embedded within a socket address, for example:

```
#include <arpa/inet.h>

// ...

char buffer[INET4_ADDRSTRLEN];
const char* result=inet_ntop(AF_INET,&ipv4addr,buffer,sizeof(buffer));
if (result==0) {
    die("failed to convert address to string (errno=%d)",errno);
}
```

IPv6 addresses can be handled by specifying AF_INET6 as the first argument, but (unlike `getnameinfo`) the result will not include the scope of a link-local or site-local address.

For both IPv4 and IPv6 the address passed in must be in network byte order (most significant byte first).

Using inet_ntoa

Another alternative is to use the function `inet_ntoa`. As with `inet_ntop`, the given IP address need not be embedded within a socket address:

```
#include <arpa/inet.h>

// ...

const char* result=inet_ntoa(&ipv4addr);
```

Notable disadvantages of `inet_ntoa` are that it is not thread safe and provides no support for IPv6. However it does pre-date both `getnameinfo` and `inet_ntop`, so is more likely to be available on older systems.

Tags: c | posix