Network Programming Laboratory
Winter 2018/2019

Session 3:
Dealing with Addressing and Naming

Janne Riihijärvi

RWTH Aachen University - Institute for Networked Systems

Contents

1 Overview

In this session we continue some time the work on the webserver assignment, with the objective of finishing the simple sequential webserver. If you have already finished this assignment last time, you can work briefly on the extensions outlined in the previous session description.

The next topic covered during the latter half of the present session is how to make socket programs portable across different versions of the IP protocol, and how to access the DNS service for converting between IP addresses and names.

2 IPv6: The Next Generation Internet Protocol

IPv6 (or Internet Protocol version 6) is the next generation Internet Protocol intended to replace the currently predominant IPv4. IPv6 is a rather conservative extension of IPv4, retaining the same basic service model and runtime operation. Perhaps the most significant change is the drastically enlarged address space: IPv6 addresses are 128 bits long compared to "only" 32 bits of IPv4. The shortage of available IPv4 addresses is one of the driving forces in the slowly ongoing transition from IPv4 towards IPv6. The new protocol version also enhances other capabilities of IP, such as autoconfiguration capabilities, multicast support, mobility support, and security functionalities. However, in this laboratory course the use of different address format will be the most prominent change.

It is also very important to note that IPv4 and IPv6 are expected to coexist for a very long time. The amount of deployed systems that only support IPv4 is still large, and will remain so for quite some time. Similarly, many networks, services and websites do not yet support IPv6 access, although the support has been rapidly improving over the past years. Operating system support for IPv6 has reached a very mature level, and most programming environments support the new IP protocol version without difficulties. Therefore, making all new network software compatible with both IP versions is highly recommended.

It is also prudent to retain support for IPv4 throughout the transition period unless there is a strong reason to do otherwise. For example, depending on the configuration of routers between a source and a destination, a native IPv6 connectivity might or might not be available. If native IPv6 support is not available, various *tunneling solutions* might still enable transport of IPv6 packets encapsulated within IPv4 ones as payload.

Programs using the socket interface can make explicit choices between the protocols used through giving the appropriate arguments to the `socket()` call, and utilizing correct socket address structures in the subsequent function calls. In the previous laboratory sessions IPv4 was explicitly used in practically all solutions. We shall see below how to support IPv6 in such explicit manner as well. Finally, we shall see how to write code in a more portable manner, without making explicit references to the IP version used, letting suitable system calls find out automatically what is supported for the desired connection.

3 IPv6 Through Sockets and Beyond

In this section we briefly outline how to get programs based on IPv4 to work with IPv6, and then how to make programs independent of the used network layer protocol altogether.

3.1 Translating IPv4 Programs to IPv6

We shall first see how to write programs using IPv6 in an explicit fashion, as was done for IPv4 in the previous sessions. From what we have learned so far, for writing code for IPv4 one has to fill in a protocol-specific structure. We have already discussed this structure for the IPv4 case, and seen how to initialize it. Recall that the key definitions are (from `man 7 ip`):

```
struct sockaddr_in {
    sa_family_t    sin_family; /* address family: AF_INET */
    in_port_t      sin_port;   /* port in network byte order */
    struct in_addr sin_addr;   /* internet address */
};

/* Internet address. */
struct in_addr {
    uint32_t       s_addr;     /* address in network byte order */
};
```

For IPv6 things are also quite similar. One can start by looking at the corresponding man page (`man 7 ipv6`). In that man page notice especially the way to define a loopback address. It will also be useful to look in to the library files, as we discussed in previous lab (`man 1 grep`), and search for the definition of `IN6ADDR_LOOPBACK_INIT`. The definition of the IPv6 address structure in general is

```
struct sockaddr_in6 {
    sa_family_t     sin6_family;   /* AF_INET6 */
    in_port_t       sin6_port;     /* port number */
    uint32_t        sin6_flowinfo; /* IPv6 flow information */
    struct in6_addr sin6_addr;     /* IPv6 address */
    uint32_t        sin6_scope_id; /* Scope ID (new in 2.4) */
};

struct in6_addr {
    unsigned char   s6_addr[16];   /* IPv6 address */
};
```

With suitable declarations and typecast this structure can be used completely analogously to `sockaddr_in` in any function call that expects a `struct sockaddr` argument. Useful functions to further read on are `inet_pton(3)`,`inet_ntop(3)`.

Please also remember that by defining a conditional macro you can check if the current host system supports or not IPv6 and if does not can just continue with IPv4 (include some warning for the user).

```
#if expression
controlled text
#endif /* expression */
```

We will distribute and discuss a figure during the lab session illustrating the differences of different socket types usually used for socket programming for the IPv4, IPv6 and the generic storage socket structure.

3.2 Using getaddrinfo() for Portability

The `getaddrinfo()` function helps for initializing the structure that works for both IPv4 and IPv6. This function takes three inputs and returns on the fourth parameter `**res` the filled in structures. The precise function prototype reads

```
int getaddrinfo(const char *node, const char *service,
                      const struct addrinfo *hints,
                      struct addrinfo **res);
```

The first two inputs are the host name and the port number, while the third one is of type `struct addrinfo *`. More information and discussion can be found on the man page `man 3 getaddrinfo`. After filling in the necessary field one can just call the function as

```
status = getaddrinfo("myhost", "3456", &addrinfo_struct, &myresults)
```

(Hint: be very careful with the fourth argument).

When trying to familiarize yourselves with the function try to understand the two following

1. What are the differences with the method we have used so far?
2. What this function returns and how one can use those for the subsequent calls to `socket()` and `bind()`? (See examples section of the man page.)

3.3 getnameinfo(), getpeername() and gethostname()

Two more important naming related functions not covered by calls already introduced you might want to have in your arsenal are

```
int getpeername(int sockfd,
                       struct sockaddr *addr, socklen_t *addrlen);
```

and

```
int gethostname(char *name, size_t len);
```

The `getpeername()` that will return you who is connected at the other end of the socket and the `gethostname()` that returns the name of your computer that the program runs. Finally, for finding out the DNS names associated to addresses `getnameinfo()` offers a protocol-independent approach that is highly recommended instead of `gethostbyaddr()` and other older varieties.

4 Programming Task

In addition to finalizing and possibly extending the webserver assignment as discussed in the introduction, do the following:

1. Make a version of the webserver code, that supports explicitly IPv6 only, like the previous version supported IPv4. Test that your server works by entering the corresponding IPv6 address to the address line of your browser.

2. Make then the server code IP-version agnostic by using `getaddrinfo` instead of explicit manipulation of address structures.

3. Finally, add code to print out the DNS name of the client as well as the server itself whenever a new connection is accepted.