Networked Systems (H) Laboratory Exercise 4: UDP/IP Networking in C

Dr Colin Perkins
School of Computing Science
University of Glasgow
https://csperkins.org/teaching/2017-2018/networked-systems/

8 March 2018

Introduction

The user datagram protocol (UDP) provides an unreliable and connectionless datagram service to applications. It is primarily used by local-area request-response protocols such as the DNS, or for applications such as voice-over-IP that prefer timeliness to reliability. UDP behaviour stands in sharp contrast to TCP, which provides a reliable, connection oriented, stream abstraction.

This exercise will explore basic UDP/IP socket programming in C. **It is a formative exercise and is not assessed.** You do not need to submit your solutions.

Background: UDP Socket Programming

A UDP socket can be created using the `socket()` system call in the usual manner, but specifying `SOCK_DGRAM` as the socket type:

```
int fd = socket(AF_INET, SOCK_DGRAM, 0);
```

Once a UDP socket has been created, it should be bound to a known port if it is expected to act as a server that sends and receives datagrams. This is done using the `bind()` system call, in exactly the same was as for a TCP socket. The arguments to `bind()` indicate the local address and port to which the socket should be bound. Since UDP is connectionless, there is no need to call the `listen()`, `accept()`, or `connect()` functions.

UDP datagrams can be transmitted using the `sendto()` function. This takes as arguments the file descriptor representing the socket, a buffer of data to be transmitted, the length of that buffer, and the addresses and port to which that buffer should be sent. The destination address is specified as a `struct sockaddr *`, and a corresponding size. Since the destination address is specified in the `sendto()` call, it's possible to send each datagram to a different destination even though they're all sent from the same socket.

```
int                fd;
char               buffer[...];
int                buflen = sizeof(buffer);
struct sockaddr_in addr = ...;
...
if (sendto(fd, buffer, buflen, 0, (struct sockaddr *) &addr, sizeof(addr)) < 0) {
  // Error...
}
```

If you have a DNS name for the host you want to send to, you need to look-up the corresponding IP address using `getaddrinfo()` in much the same way as you would for a TCP socket, and pass the IP address to `sendto()`. If `getaddrinfo()` returns several IP addresses, you should probe them to find out which addresses are usable. Unlike a TCP socket, where you can just try to `connect()` to each in turn, you must implement the probing yourself, with your application code sending the probe messages and replies, and handling timeouts.

The `recvfrom()` function can be used to receive UDP datagrams. This works in much the same way as `read()`, except that it also takes an empty address structure (`struct sockaddr *`) that is filled in with source address and port from the received datagram. This address can be used in a `sendto()` call to send a reply. Each received datagram can come from a different source address.

```
int             fd;
char            buffer[...];
int             buflen = sizeof(buffer);
struct sockaddr addr;
socklen_t       alen = sizeof(addr);
int             rlen;
...
rlen = recvfrom(fd, buffer, buflen, 0, &addr, &alen);
if (rlen < 0) {
  // Error...
}
```

A UDP socket must be closed in the usual way, once you have finished using it, using the `close()` system call.

Formative Exercise 3: UDP Client/Server Example

The formative exercise for this lab demonstrates how to build the most simple UDP-based client-server application. You should write two programs:

`udp_hello_server` The server should listen for datagrams on UDP port 5008. It should read the first datagram received, print the contents of that datagram and the IP address from which it came to the screen, close the socket, then exit. You can use the `inet_ntop()` function to print an IP address.

`udp_hello_client` Your client should send the text "Hello, world!" in a UDP datagram to port 5008 of a host named on the command line, then it should close the socket. The client should take the name of the machine on which the server is running as its single command line argument (i.e., if the server is running on machine `bo720-1-01` you should run your client using the command `udp_hello_client bo720-1-01`. For the purposes of this exercise, send the UDP datagram to the first address that `getaddrinfo()` returns.

Run your client and server, and demonstrate that you can send the text "Hello, world!" from one to the other. Try this with client and server running on the same machine, and with them running on two different machines.

- + -