VE489 Computer Networks

Socket Programming

Author: Qinye Li

Date: June 16, 2017

What is a socket?

Why socket?

- telnet
- wget
- curl

Types of Sockets

- Stream socket
  - Uses TCP
- Datagram socket
  - Uses UDP
- And other sockets that you don't need to know for now

How to use socket?

`socket()` -- returns the file descriptor

```c
// Create a stream socket (SOCK_STREAM)
// over the internet (AF_INET)
int sock = socket(AF_INET, SOCK_STREAM, 0);
```

```c
// Create a datagram socket (SOCK_DRAM)
// over the internet (AF_INET)
int sock = socket(AF_INET, SOCK_DGRAM, 0);
```

`sockaddr_in` -- describes the address

Socket describes

- protocol

Socket address describes

- ip adderess
- port number

`sockaddr_in`

```
sockaddr_in addr;
addr.sin_family = AF_INET;
addr.sin_port = htons(port);
// port number must be in the range 1024 to 65535

if (isServer) {
    addr.sin_addr.s_addr = htonl(INADDR_ANY);
    // INADDR_ANY means any connection

} else if (isClient) {
    addr.sin_addr.s_addr = inet_addr(hostIP.c_str());
}
```

The Network Order

- General computer byte order: little-endian
- Network byte order: big-endian

```
htonl(); // host ot network long
htons(); // host to network short
ntohl(); // network to host long
ntohs(); // netowrk to host short
```

more specificly,

```
uint32_t htonl(uint32_t hostlong);
uint16_t htons(uint16_t hostshort);
uint32_t ntohl(uint32_t netlong);
uint16_t ntohs(uint16_t netshort);
```

`bind()`

```
// Request the port from os
// If port = -1, os will randomly assign a port
::bind(sock, (sockaddr*)&addr, sizeof(addr));
```

`connect()`

```
connect(sock, (sockaddr*)&addr, sizeof(addr));
```

`listen()` and `accept()`

```c
listen(sock, 0);
int clientSock = accept(sock, 0, 0);
```

`send()` and `recv()`

```
if (isClient) {
    connection = sock;

} elseif (isServer) {
    connection = clientSock;
}
```

`send()` and `recv()`

```
string message;
send(connection, message.c_str(), message.size(), 0);
// returns the number of bytes actually sent
```

```
char buf[BUFFER_SIZE];
recv(connection, buf, BUFFER_SIZE, 0); // blocking
// returns the number of bytes actually received
// If returns 0, the remote side has closed connection
```

`close()`

```
close(connection);
```

Code it defensively!

```
if (::bind(...)) {
        cerr << strerror(errno) << endl;
}
```

```
if (listen(...)) {
        cerr << strerror(errno) << endl;
}
```

```
if (connect(...)) {
        cerr << strerror(errno) << endl;
}
```

Code it defensively!

```
int bytesReceived = 0;
while(bytesReceived <= bytesExpected) {
    int rc = recv(connection, buf, BUFFER_SIZE, 0);
    if (rc == -1) {
        cerr << strerror(errno) << endl;
    }
    bytesReceived += rc;
}
```

```
recv(connection, buf, BUFFER_SIZE, MSG_WAITALL);
```

Demo

Talk to me the Datagram style ; -)

- `listen()`, `connect()` and `accept()` no longer needed

`sendto()` and `recvfrom()`

```
string message;
sendto(sock, message.c_str(), message.size(), 0,
         (sockaddr*) &addr, sizeof(sockaddr_in));
```

```
char buf[BUFFER_SIZE];
recvfrom(sock, buf, BUFFER_SIZE, 0,
         (sockaddr*) &addr, sizeof(sockaddr_in));
```

Things not mentioned in class

Here are a couple of things I didn't mention in class.

Well, I didn't want to overwhelm you with the complexity and distract you from what really matters. So I shaved off those complicated pieces in my demo code.

Sure, sure, you can use them in your code. They work. No one is going to hit you for this. But they are just not the best practise.

Now that you are curious enough to get back to these slides, I am going to talk about the better practise.

`sockaddr_in` and `sockaddr_in6`

The `sockaddr_in` in page 9 is for IPv4 specificly. The IPv6 version is called `sockaddr_in6`. Of course, there is away to code in a way that's compatible with both IPv4 and IPv6. No need to manually packs the `struct sockaddr_in` before calling `bind()` in this case.

See Beej's Guid Section 5.1 `getaddrinfo()` and Section 5.3 bind() for more detail.

Conversion between `string` and `struct in_addr` ip address

I used `inet_addr()` in page 8 to convert `string` ip address to `struct in_addr` ip address. Yet this function is for IPv4 specificly, and is deprecated even for IPv4 use. But why do I use it here? Cuz it's got the easiest interface =).

Ok. Then what's the better approach?

```
inet_pton(); // printable to network ip address
inet_ntop(); // network to printable ip address
```

More specificly,

```
const char *inet_ntop(int af, const void *src,
                      char *dst, socklen_t size);

int inet_pton(int af, const char *src, void *dst);
```

Confused?

Don't worry, these are on the "I can always look them up when I need" list. Just remember this is used for conversion between human-readable and computer-readable ip address.

See Beej's Guid Section 9.13 `inet_ntoa()` and Section 9.14 `inet_ntop()` for more detail.

Ref

[1] Wikipeadia -- Network socket

Wikipeadia can be a nice starting point for you to explore a new concept!

[2] Beej's Guide to Network Programming

Explains socket programming in a beginner-friendly way. Helped me a LOT when I was doing my 489 projects. Recommended by my EECS489 instructor.