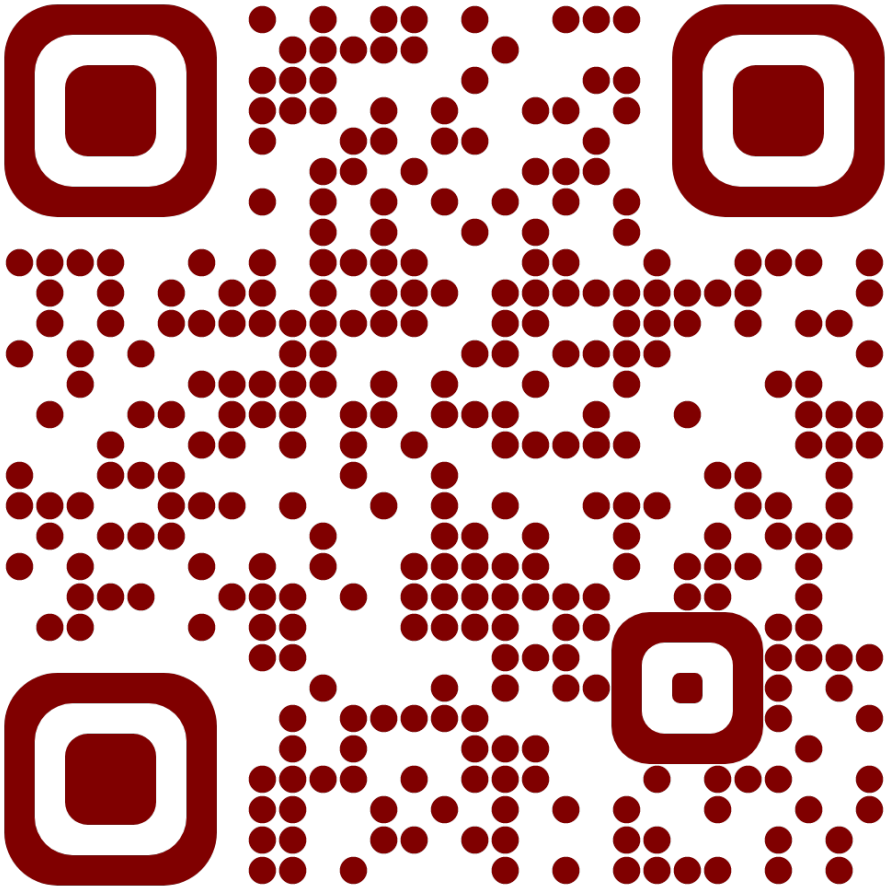
Socket Programming

Rathindra Nath Dutta

Senior Research Fellow
Advanced Computing & Microelectronics Unit
Indian Statistical Institute, Kolkata

October 28, 2022

https://ratcoinc.github.io/Networks/



WEB PAGE

Displaying Client Info

- A typical connection accept mechanism looks like:

```
struct sockaddr_in cli_addr;
int cli_addr_len = sizeof(cli_addr);
int accepted_sockfd = accept(sockfd,
        (struct sockaddr *)&cli_addr, &cli_addr_len);
```

Displaying Client Info

- A typical connection accept mechanism looks like:

```
struct sockaddr_in cli_addr;
int cli_addr_len = sizeof(cli_addr);
int accepted_sockfd = accept(sockfd,
        (struct sockaddr *)&cli_addr, &cli_addr_len);
```

- To display IP and Port of incoming connection:

```
printf("IP address is: %s\n", inet_ntoa(cli_addr.sin_addr));
printf("port is: %d\n", (int) ntohs(cli_addr.sin_port));
```

[1] `inet_ntoa()` converts the Internet host address given in network byte order, to a string in IPv4 dotted-decimal notation

[2] `ntohs()` converts the given unsigned short integer from network byte order to host byte order

Exercise 1

Modify the Echo Server program: `server1.c`
To Print `telnet` client's IP and Port address

[1] Use `ifconfig -a` or `ip addr` to get local IP address

[2] Use `netstat -na | grep <portno>` to get status of that port

Exercise 1

Modify the Echo Server program: `server1.c`
To Print `telnet` client's IP and Port address

Solution

Simply uncomment the lines 50 and 51 in `server1.c`

[1] Use `ifconfig -a` or `ip addr` to get local IP address

[2] Use `netstat -na | grep <portno>` to get status of that port

Creating a Client

A typical connection request mechanism:

```
int sockfd = socket(AF_INET, SOCK_STREAM, 0);

struct sockaddr_in serv_addr;
bzero((char *) &serv_addr, sizeof(serv_addr));
serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr=inet_addr("127.0.0.1"); //server IP
serv_addr.sin_port = htons(54321); // server port

connect(sockfd, (struct sockaddr *)&serv_addr,
            sizeof(serv_addr))

// read/write using the socket
```

Testing the Client Program

Complete programs: `client1.c`, `server1.c`

open a terminal for server process
`gcc server1.c -o server && ./server`
leave this window open

open another terminal for client process
`gcc client1.c -o client && ./client`
send "quit" to stop

Sending an Integer over a Socket

- Socket communication uses byte stream
- Integer (or anything) needs to interpreted as raw bytes[1]
- Always write machine independent codes:
  use `htonl()`, `ntohl()`, `uint32_t` or similar things[2]

[1] The process is known as **Serialization**/**Deserialization**

[2] See this discussion: `https://stackoverflow.com/questions/9140409/transfer-integer-over-a-socket-in-c`

Sending an Integer over a Socket

- Socket communication uses byte stream
- Integer (or anything) needs to interpreted as raw bytes[1]
- Always write machine independent codes:
  use `htonl()`, `ntohl()`, `uint32_t` or similar things[2]

```
uint32_t number_to_send = 10000; // Put your value
uint32_t converted_num = htonl(number_to_send);
write(socketfd, &converted_num, sizeof(uint32_t));
```

[1] The process is known as **Serialization**/**Deserialization**

[2] See this discussion: `https://stackoverflow.com/questions/9140409/transfer-integer-over-a-socket-in-c`

Sending an Integer over a Socket

- Socket communication uses byte stream
- Integer (or anything) needs to interpreted as raw bytes[1]
- Always write machine independent codes:
  use `htonl()`, `ntohl()`, `uint32_t` or similar things[2]

```
uint32_t number_to_send = 10000; // Put your value
uint32_t converted_num = htonl(number_to_send);
write(socketfd, &converted_num, sizeof(uint32_t));

uint32_t read_num, num;
read(socketfd, &read_num, sizeof(uint32_t));
num = ntohl(read_num); // get the actual value
```

[1] The process is known as **Serialization**/**Deserialization**

[2] See this discussion: `https://stackoverflow.com/questions/9140409/transfer-integer-over-a-socket-in-c`

Sending an Integer over a Socket

- Socket communication uses byte stream
- Integer (or anything) needs to interpreted as raw bytes[1]
- Always write machine independent codes:
  use `htonl()`, `ntohl()`, `uint32_t` or similar things[2]

```
uint32_t number_to_send = 10000; // Put your value
uint32_t converted_num = htonl(number_to_send);
write(socketfd, &converted_num, sizeof(uint32_t));

uint32_t read_num, num;
read(socketfd, &read_num, sizeof(uint32_t));
num = ntohl(read_num); // get the actual value
```

- Other types (e.g. `float`[3][4]) can also be sent in similar fashion

[1] The process is known as **Serialization**/**Deserialization**

[2] See this discussion: `https://stackoverflow.com/questions/9140409/transfer-integer-over-a-socket-in-c`

[3] Sending float: `https://stackoverflow.com/questions/38511305/sending-float-values-on-socket-c-c`

[4] Serialization (How to Pack Data): `https://beej.us/guide/bgnet/html/#serialization`

Exercise 2

Modify both the Server and Client programs: `server1.c`, `client1.c`
send an integer n (32 bit) to the server and
gets an integer $f(n)$ from the server
for now, assume $f(n) = n + 1$

Exercise 2

Modify both the Server and Client programs: `server1.c`, `client1.c`
send an integer n (32 bit) to the server and
gets an integer $f(n)$ from the server
for now, assume $f(n) = n + 1$

Solution

Take a look at: `server2.c`, `client2.c`

Accepting Multiple Clients

uisng `fork()`:

`https://github.com/kusdavletov/socket-programming-simple-server-and-client/blob/master/server.c`

using `pthread`: `https://www.geeksforgeeks.org/`

`handling-multiple-clients-on-server-with-multithreading-using-socket-programming-in-c-cpp/`

Exercise 3

Write a multi-client Server program to which each Client (at least 2) sends an integer n (32 bit) and the server returns the integer $n + 1$

Exercise 3

Write a multi-client Server program to which each Client (at least 2) sends an integer n (32 bit) and the server returns the integer $n + 1$

Solution

Take a look at: `server3.c`