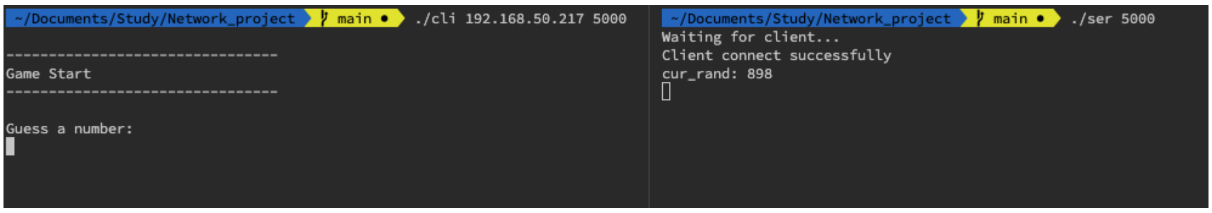
## 1. Game Demo

(1) connect (client | server)

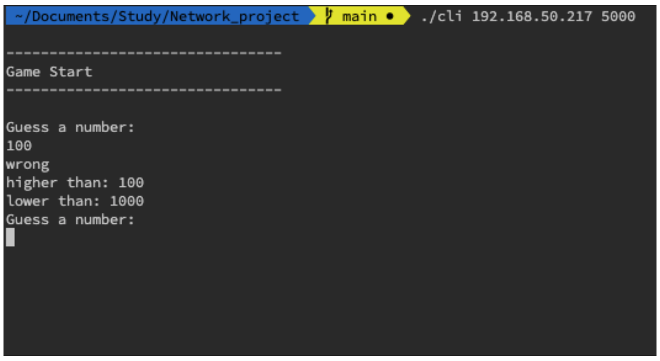
`cur_rand` is the random number generated by the server for this round.



```
~/Documents/Study/Network_project  main ●  ./cli 192.168.50.217 5000

--------------------------------
Game Start
--------------------------------

Guess a number:
```

```
~/Documents/Study/Network_project  main ●  ./ser 5000
Waiting for client...
Client connect successfully
cur_rand: 898
```

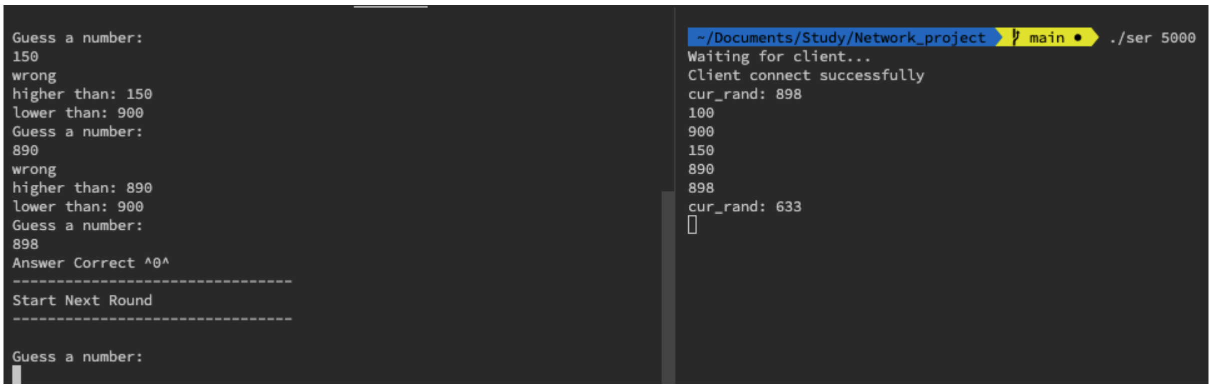
(2) after first guess(client side)



```
~/Documents/Study/Network_project  main ●  ./cli 192.168.50.217 5000

--------------------------------
Game Start
--------------------------------

Guess a number:
100
wrong
higher than: 100
lower than: 1000
Guess a number:
```

(3) after several guess, show correct

Client can directly start a new round.

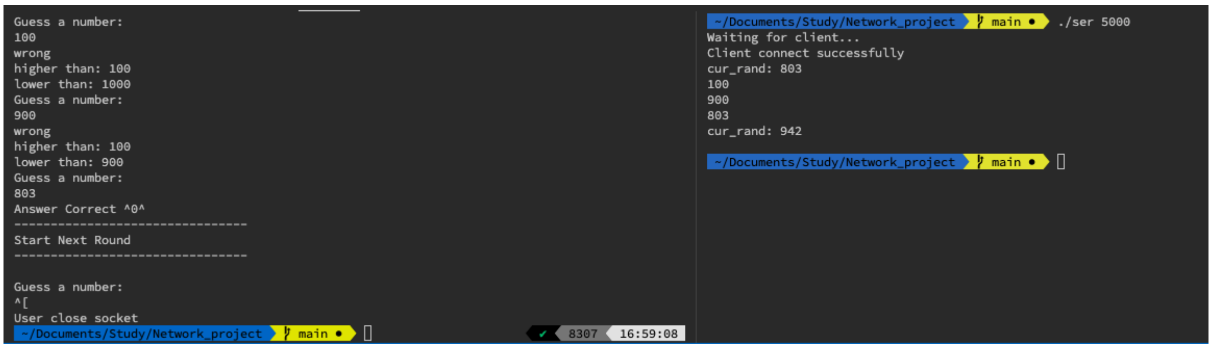


```
Guess a number:
150
wrong
higher than: 150
lower than: 900
Guess a number:
890
wrong
higher than: 890
lower than: 900
Guess a number:
898
Answer Correct ^0^
--------------------------------
Start Next Round
--------------------------------

Guess a number:
```

```
~/Documents/Study/Network_project  main ●  ./ser 5000
Waiting for client...
Client connect successfully
cur_rand: 898
100
900
150
890
898
cur_rand: 633
```

(4) press ESC to close

client can press ESC to close the socket.



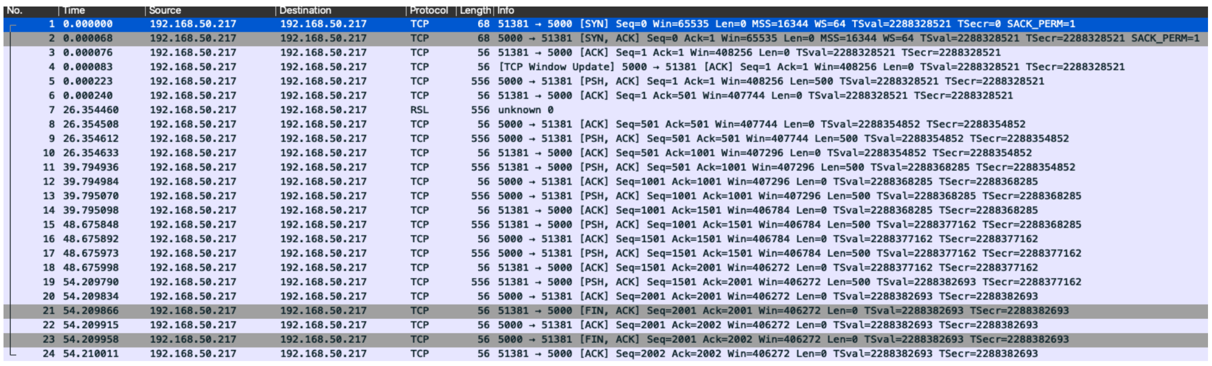
```
Guess a number:
100
wrong
higher than: 100
lower than: 1000
Guess a number:
900
wrong
higher than: 100
lower than: 900
Guess a number:
803
Answer Correct ^0^
--------------------------------
Start Next Round
--------------------------------

Guess a number:
^[
User close socket
~/Documents/Study/Network_project  main ●                ✓  8307  16:59:08
```

```
~/Documents/Study/Network_project  main ●  ./ser 5000
Waiting for client...
Client connect successfully
cur_rand: 803
100
900
803
cur_rand: 942

~/Documents/Study/Network_project  main ●
```

## 2. Wireshark

(1) TCP handshaking packets

NO.1,2,3



```
No. Time      Source          Destination     Protocol Length Info
  1 0.000000  192.168.50.217  192.168.50.217  TCP      68  51381 → 5000 [SYN] Seq=0 Win=65535 Len=0 MSS=16344 WS=64 TSval=2288328521 TSecr=0 SACK_PERM=1
  2 0.000068  192.168.50.217  192.168.50.217  TCP      68  5000 → 51381 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=16344 WS=64 TSval=2288328521 TSecr=2288328521 SACK_PERM=1
  3 0.000076  192.168.50.217  192.168.50.217  TCP      56  51381 → 5000 [ACK] Seq=1 Ack=1 Win=408256 Len=0 TSval=2288328521 TSecr=2288328521
  4 0.000083  192.168.50.217  192.168.50.217  TCP      56  [TCP Window Update] 5000 → 51381 [ACK] Seq=1 Ack=1 Win=408256 Len=0 TSval=2288328521 TSecr=2288328521
  5 0.000223  192.168.50.217  192.168.50.217  TCP     556  5000 → 51381 [PSH, ACK] Seq=1 Ack=1 Win=408256 Len=500 TSval=2288328521 TSecr=2288328521
  6 0.000240  192.168.50.217  192.168.50.217  TCP      56  51381 → 5000 [ACK] Seq=1 Ack=501 Win=407744 Len=0 TSval=2288328521 TSecr=2288328521
  7 26.354460 192.168.50.217  192.168.50.217  RSL     556  unknown 0
  8 26.354508 192.168.50.217  192.168.50.217  TCP      56  5000 → 51381 [ACK] Seq=501 Ack=501 Win=407744 Len=0 TSval=2288354852 TSecr=2288354852
  9 26.354612 192.168.50.217  192.168.50.217  TCP     556  5000 → 51381 [PSH, ACK] Seq=501 Ack=501 Win=407744 Len=500 TSval=2288354852 TSecr=2288354852
 10 26.354633 192.168.50.217  192.168.50.217  TCP      56  51381 → 5000 [ACK] Seq=501 Ack=1001 Win=407296 Len=0 TSval=2288354852 TSecr=2288354852
 11 39.794936 192.168.50.217  192.168.50.217  TCP     556  51381 → 5000 [PSH, ACK] Seq=501 Ack=1001 Win=407296 Len=500 TSval=2288368285 TSecr=2288354852
 12 39.794984 192.168.50.217  192.168.50.217  TCP      56  5000 → 51381 [ACK] Seq=1001 Ack=1001 Win=407296 Len=0 TSval=2288368285 TSecr=2288368285
 13 39.795070 192.168.50.217  192.168.50.217  TCP     556  5000 → 51381 [PSH, ACK] Seq=1001 Ack=1001 Win=407296 Len=500 TSval=2288368285 TSecr=2288368285
 14 39.795098 192.168.50.217  192.168.50.217  TCP      56  51381 → 5000 [ACK] Seq=1001 Ack=1501 Win=406784 Len=0 TSval=2288368285 TSecr=2288368285
 15 48.675848 192.168.50.217  192.168.50.217  TCP     556  51381 → 5000 [PSH, ACK] Seq=1001 Ack=1501 Win=406784 Len=500 TSval=2288377162 TSecr=2288368285
 16 48.675892 192.168.50.217  192.168.50.217  TCP      56  5000 → 51381 [ACK] Seq=1501 Ack=1501 Win=406784 Len=0 TSval=2288377162 TSecr=2288377162
 17 48.675973 192.168.50.217  192.168.50.217  TCP     556  5000 → 51381 [PSH, ACK] Seq=1501 Ack=1501 Win=406784 Len=500 TSval=2288377162 TSecr=2288377162
 18 48.675998 192.168.50.217  192.168.50.217  TCP      56  51381 → 5000 [ACK] Seq=1501 Ack=2001 Win=406272 Len=0 TSval=2288377162 TSecr=2288377162
 19 54.209790 192.168.50.217  192.168.50.217  TCP     556  51381 → 5000 [PSH, ACK] Seq=1501 Ack=2001 Win=406272 Len=500 TSval=2288382693 TSecr=2288377162
 20 54.209834 192.168.50.217  192.168.50.217  TCP      56  5000 → 51381 [ACK] Seq=2001 Ack=2001 Win=406272 Len=0 TSval=2288382693 TSecr=2288382693
 21 54.209866 192.168.50.217  192.168.50.217  TCP      56  51381 → 5000 [FIN, ACK] Seq=2001 Ack=2001 Win=406272 Len=0 TSval=2288382693 TSecr=2288382693
 22 54.209915 192.168.50.217  192.168.50.217  TCP      56  5000 → 51381 [ACK] Seq=2001 Ack=2002 Win=406272 Len=0 TSval=2288382693 TSecr=2288382693
 23 54.209958 192.168.50.217  192.168.50.217  TCP      56  5000 → 51381 [FIN, ACK] Seq=2001 Ack=2002 Win=406272 Len=0 TSval=2288382693 TSecr=2288382693
 24 54.210011 192.168.50.217  192.168.50.217  TCP      56  51381 → 5000 [ACK] Seq=2002 Ack=2002 Win=406272 Len=0 TSval=2288382693 TSecr=2288382693
```

(2) server, client IP address

192.168.50.217

(3) server, client port

client: 51381

server: 5000

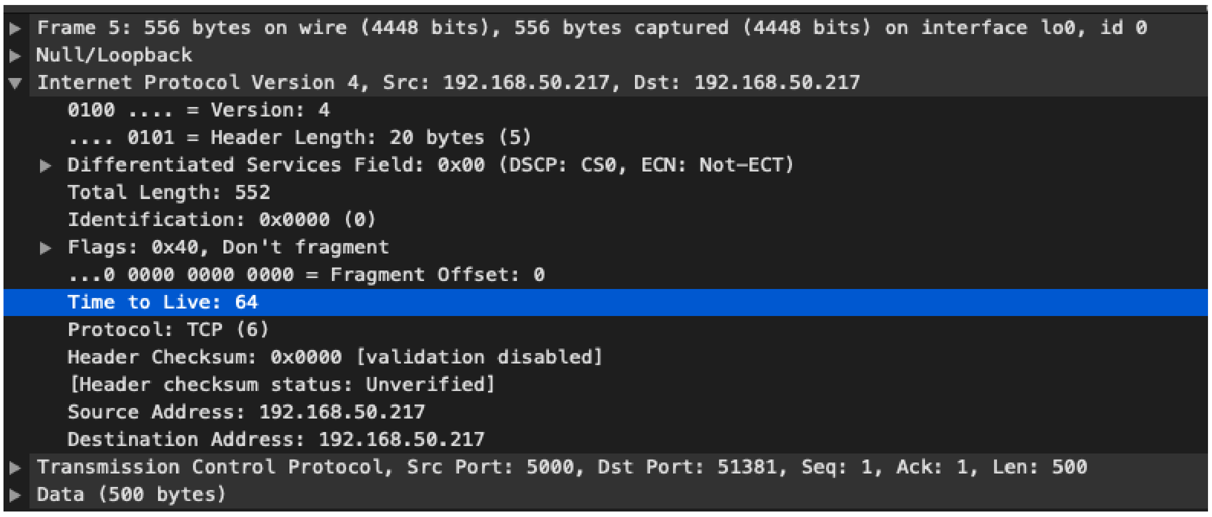
(4) Size of packet from client

556 bytes ( picture of (1) )

(5) How many routers does each of the transmitted packets go through?

0

TTL of every packet=64, which is the default TTL value of TCP for Linux and MacOS. When a packet goes through a router, TTL increases by 1. If TTL of every packet is 64, then the packet doesn't go through any routers.



```
▶ Frame 5: 556 bytes on wire (4448 bits), 556 bytes captured (4448 bits) on interface lo0, id 0
▶ Null/Loopback
▼ Internet Protocol Version 4, Src: 192.168.50.217, Dst: 192.168.50.217
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 552
    Identification: 0x0000 (0)
  ▶ Flags: 0x40, Don't fragment
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 64
    Protocol: TCP (6)
    Header Checksum: 0x0000 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 192.168.50.217
    Destination Address: 192.168.50.217
▶ Transmission Control Protocol, Src Port: 5000, Dst Port: 51381, Seq: 1, Ack: 1, Len: 500
▶ Data (500 bytes)
```

## 3. Implementation

(1) server

- call data structure and functions

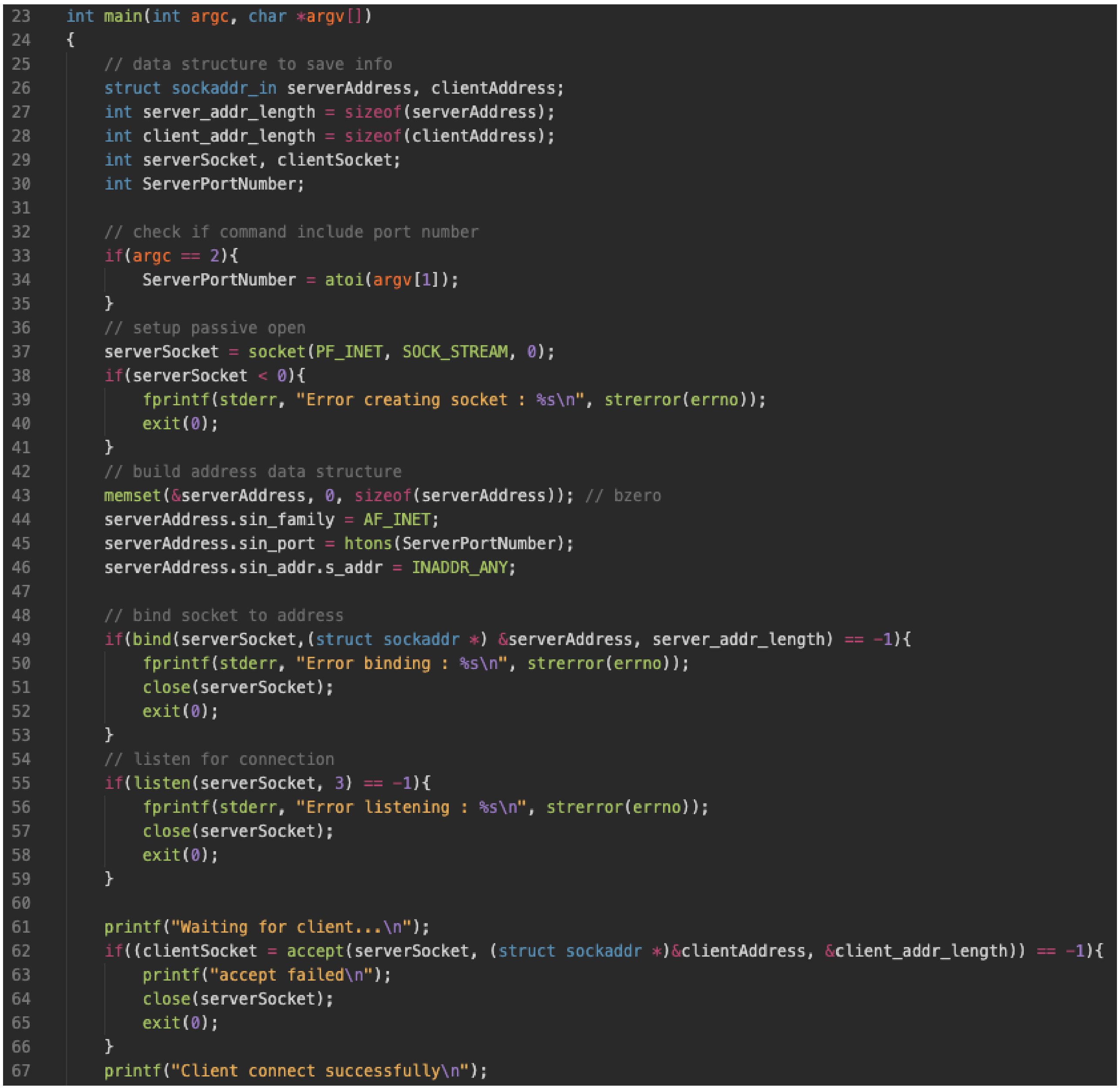
```
#define RANGE 1000 // rand range = 0 ~ 999
int cur_rand; // current rand number
int lower_bound = 0;
int upper_bound = RANGE;
int rand_gen(); // generate a random number
void update_range(int a); // update bound by client input
```

```
int serverSocket, clientSocket;
```

- functions

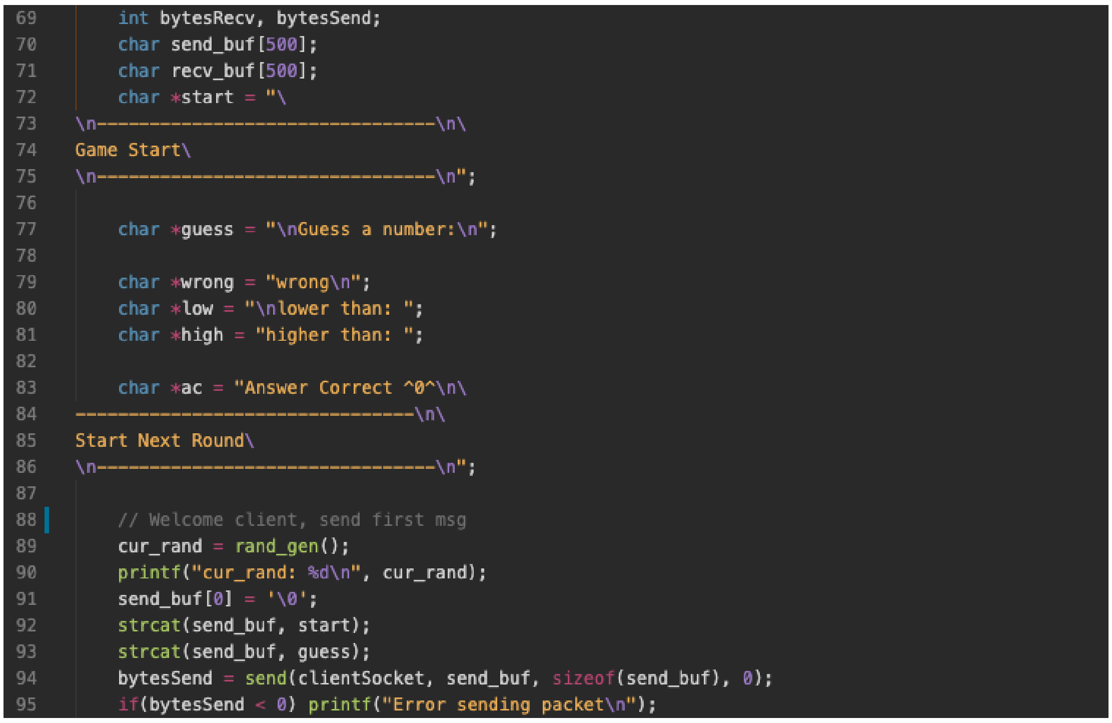
```
// return a random number ranged from 0 to RANGE-1
int rand_gen()
{
   time_t t;
   srand((unsigned) time(&t));
   return rand() % RANGE; // RANGE 1000
}

// update upper_bound and lower_bound by a(client input)
void update_range(int a)
{
   if(a < cur_rand && a > lower_bound) lower_bound = a;
   if(a > cur_rand && a < upper_bound) upper_bound = a;
}
```

- main function
    - connection



```c
int main(int argc, char *argv[])
{
    // data structure to save info
    struct sockaddr_in serverAddress, clientAddress;
    int server_addr_length = sizeof(serverAddress);
    int client_addr_length = sizeof(clientAddress);
    int serverSocket, clientSocket;
    int ServerPortNumber;

    // check if command include port number
    if(argc == 2){
        ServerPortNumber = atoi(argv[1]);
    }
    // setup passive open
    serverSocket = socket(PF_INET, SOCK_STREAM, 0);
    if(serverSocket < 0){
        fprintf(stderr, "Error creating socket : %s\n", strerror(errno));
        exit(0);
    }
    // build address data structure
    memset(&serverAddress, 0, sizeof(serverAddress)); // bzero
    serverAddress.sin_family = AF_INET;
    serverAddress.sin_port = htons(ServerPortNumber);
    serverAddress.sin_addr.s_addr = INADDR_ANY;

    // bind socket to address
    if(bind(serverSocket,(struct sockaddr *) &serverAddress, server_addr_length) == -1){
        fprintf(stderr, "Error binding : %s\n", strerror(errno));
        close(serverSocket);
        exit(0);
    }
    // listen for connection
    if(listen(serverSocket, 3) == -1){
        fprintf(stderr, "Error listening : %s\n", strerror(errno));
        close(serverSocket);
        exit(0);
    }

    printf("Waiting for client...\n");
    if((clientSocket = accept(serverSocket, (struct sockaddr *)&clientAddress, &client_addr_length)) == -1){
        printf("accept failed\n");
        close(serverSocket);
        exit(0);
    }
    printf("Client connect successfully\n");
```

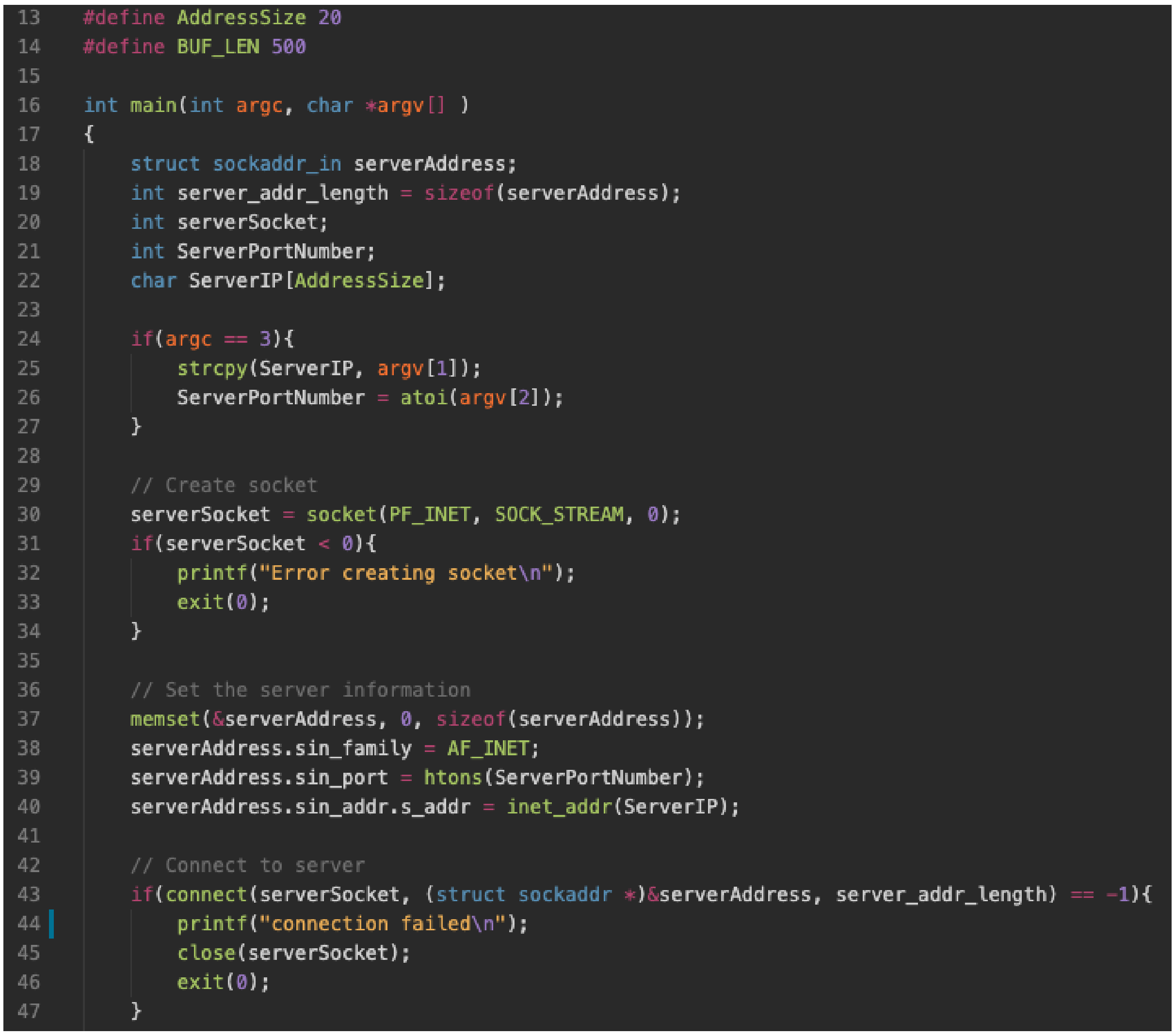
- play
    - send Game Start message



```c
    int bytesRecv, bytesSend;
    char send_buf[500];
    char recv_buf[500];
    char *start = "\
\n--------------------------------\n\
Game Start\
\n--------------------------------\n";

    char *guess = "\nGuess a number:\n";

    char *wrong = "wrong\n";
    char *low = "\nlower than: ";
    char *high = "higher than: ";

    char *ac = "Answer Correct ^0^\n\
--------------------------------\n\
Start Next Round\
\n--------------------------------\n";

    // Welcome client, send first msg
    cur_rand = rand_gen();
    printf("cur_rand: %d\n", cur_rand);
    send_buf[0] = '\0';
    strcat(send_buf, start);
    strcat(send_buf, guess);
    bytesSend = send(clientSocket, send_buf, sizeof(send_buf), 0);
    if(bytesSend < 0) printf("Error sending packet\n");
```

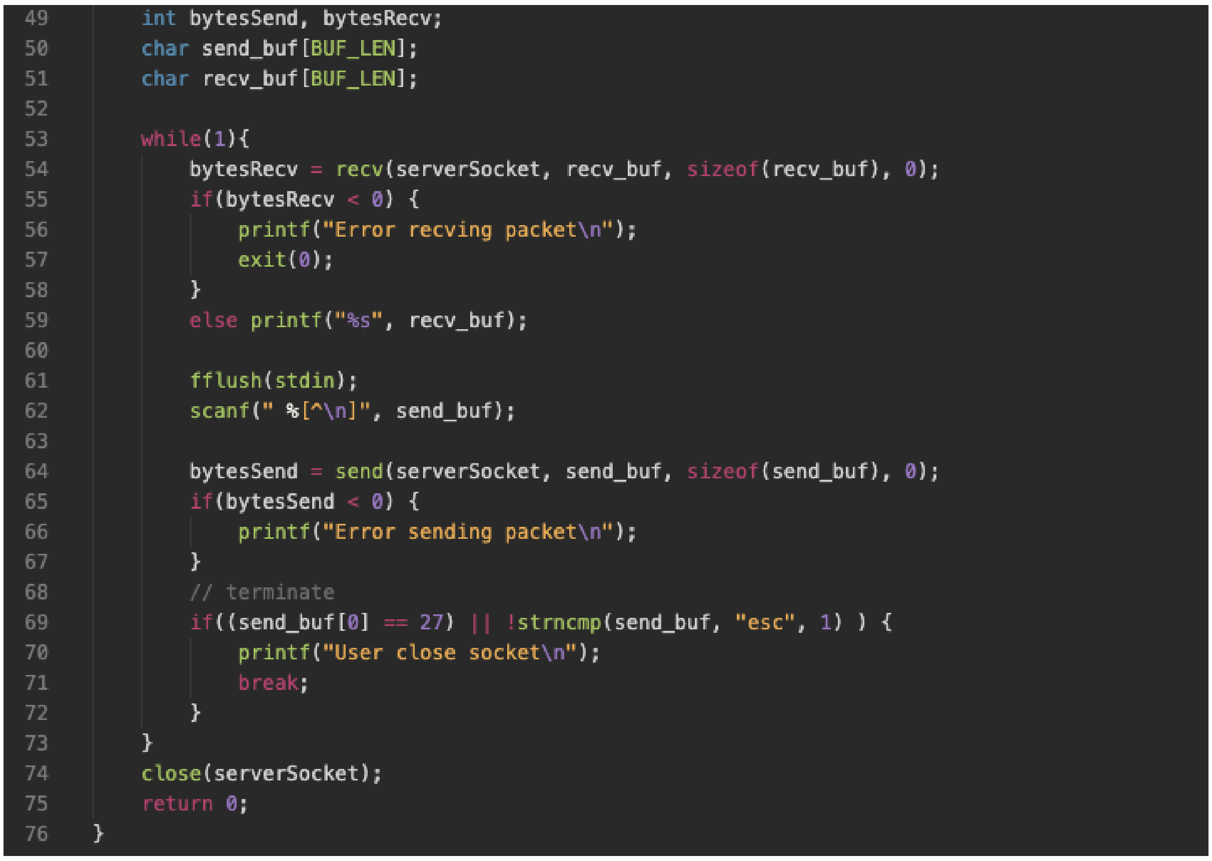
(2) client

- connection



```c
#define AddressSize 20
#define BUF_LEN 500

int main(int argc, char *argv[] )
{
    struct sockaddr_in serverAddress;
    int server_addr_length = sizeof(serverAddress);
    int serverSocket;
    int ServerPortNumber;
    char ServerIP[AddressSize];

    if(argc == 3){
        strcpy(ServerIP, argv[1]);
        ServerPortNumber = atoi(argv[2]);
    }

    // Create socket
    serverSocket = socket(PF_INET, SOCK_STREAM, 0);
    if(serverSocket < 0){
        printf("Error creating socket\n");
        exit(0);
    }

    // Set the server information
    memset(&serverAddress, 0, sizeof(serverAddress));
    serverAddress.sin_family = AF_INET;
    serverAddress.sin_port = htons(ServerPortNumber);
    serverAddress.sin_addr.s_addr = inet_addr(ServerIP);

    // Connect to server
    if(connect(serverSocket, (struct sockaddr *)&serverAddress, server_addr_length) == -1){
        printf("connection failed\n");
        close(serverSocket);
        exit(0);
    }
```

- play
    - `recv` get packet
    - `send` send user input packet
    - if user input ESC, close socket connection



```c
    int bytesSend, bytesRecv;
    char send_buf[BUF_LEN];
    char recv_buf[BUF_LEN];

    while(1){
        bytesRecv = recv(serverSocket, recv_buf, sizeof(recv_buf), 0);
        if(bytesRecv < 0) {
            printf("Error recving packet\n");
            exit(0);
        }
        else printf("%s", recv_buf);

        fflush(stdin);
        scanf(" %[^\n]", send_buf);

        bytesSend = send(serverSocket, send_buf, sizeof(send_buf), 0);
        if(bytesSend < 0) {
            printf("Error sending packet\n");
        }
        // terminate
        if((send_buf[0] == 27) || !strncmp(send_buf, "esc", 1) ) {
            printf("User close socket\n");
            break;
        }
    }
    close(serverSocket);
    return 0;
}
```