CENG393 Computer Networks
Labwork 3

Socket Programming: TCP and UDP Sockets

1 Socket Programming

A socket is an interprocess communication mechanism for establishing communication between programs over computer networks. In order to establish a working socket, the following information has to be provided in the code:

- Socket domain (local, IPv4, IPv6, etc)
- Socket type (connection oriented, connectionless, raw, etc)
- Port number (1 - 65535)
- Destination protocol address

In our laboratories, we will focus on network sockets over IPv4.

1.1 TCP Sockets

TCP (Transmission Control Protocol) is a standard transport layer protocol that provides connection oriented, reliable and error-free data communication [1]. The process of establishing and utilizing a connection oriented TCP socket is demonstrated in Figure 1:

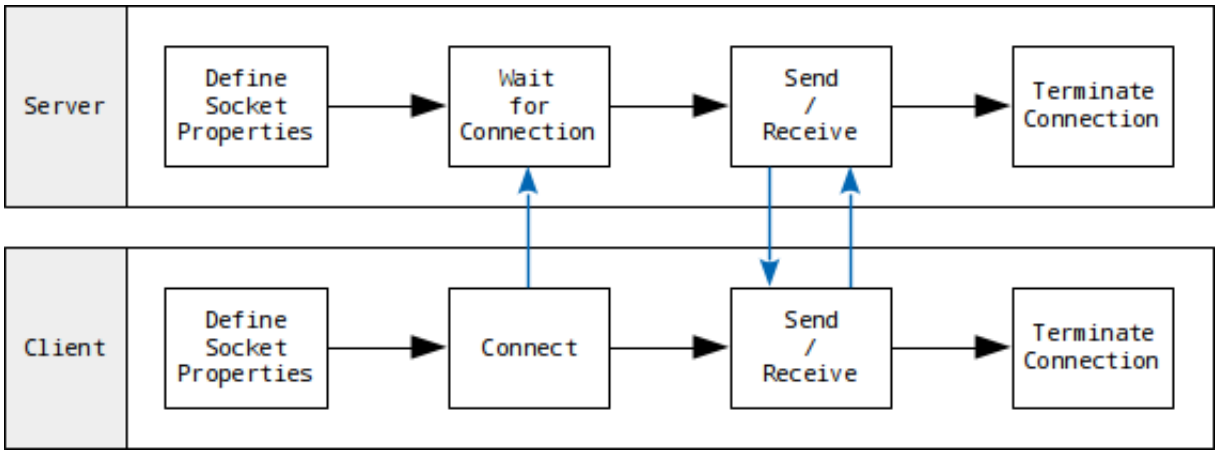

Figure 1: TCP Sockets

1.2 UDP Sockets

UDP (User Datagram Protocol) is an earlier transport layer protocol, which has less overhead from TCP [2]. Remember that TCP requires establishment of a connection first and uses acknowledgement messages and sequence numbers to guarentee that messages are delivered correctly to the other end of the socket. In contrary to TCP, UDP utilizes none of these methods: it just sends messages without acknowledging whether the message has been received by the other end of the socket or not. It is a faster way of transmitting data across the network but it is more vulnerable to message loss and corruption (therefore correction must be handled by upper-layer application protocols).

The process of establishing and utilizing a connectionless UDP socket is demonstrated in Figure 2:

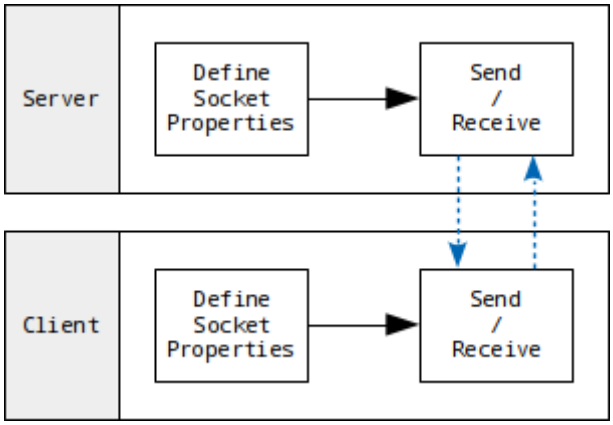

Figure 2: UDP Sockets

2 Exercises

1. Study manpages of the following library functions and system calls: **socket setsockopt htons bind listen connect accept send recv close**.

2. Study manpages of **sendto** and **recvfrom** system calls. What is the difference of **send** from **sendto** and **recv** from **recvfrom**?

3. The sample codes in Appendices A and B uses TCP sockets to send a message to a client from a server. Modify the codes to send more messages and receive replies from the client. The program should terminate when the client sends the message "END".

4. The sample codes in Appendics C and D uses UDP sockets to send messages from client to server in a loop. Modify both codes such that server keeps a list of blacklisted IP addresses and sends back "MESSAGE REJECTED" as reply if a message has been received from a blacklisted client.

Appendices

A tcpserver.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/wait.h>

#define MYPORT 3490 // the port users will be connecting to
#define BACKLOG 10 // how many pending connections queue will hold

int main() {
  int sockfd, new_fd;
  struct sockaddr_in my_addr; // my address information
  struct sockaddr_in their_addr; // connector's address information
  int sin_size;
  int yes = 1;

  if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
    perror("socket");
    exit(1);
  }

  setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(int));
  my_addr.sin_family = AF_INET; //Internet Address Family (IP)
  my_addr.sin_port = htons(MYPORT); // short, network byte order
  my_addr.sin_addr.s_addr = INADDR_ANY;
  memset(&(my_addr.sin_zero), '\0', 8);

  if (bind(sockfd, (struct sockaddr *) &my_addr, sizeof(struct sockaddr)) == -1) {
    perror("bind");
    exit(1);
  }

  if (listen(sockfd, BACKLOG) == -1) {
    perror("listen");
    exit(1);
  }

  sin_size = sizeof(struct sockaddr_in);
  if ((new_fd = accept(sockfd, (struct sockaddr *) &their_addr, &sin_size)) == -1) {
    perror("accept");
    exit(1);
  }

  if (send(new_fd, "Hello world!\n", 14, 0) == -1) {
    perror("send");
    close(new_fd);
    exit(0);
  }

  return 0;
}
```

B tcpclient.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <netdb.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <arpa/inet.h>

#define PORT 3490
#define MAXDATASIZE 100 // max number of bytes we can get at once

int main() {
  char IP[16];
  int sockfd, numbytes;
  char buf[MAXDATASIZE];
  struct sockaddr_in server; // connector's address information

  if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
    perror("socket");
    exit(1);
  }

  server.sin_family = AF_INET; // Internet Address Family (IP)
  server.sin_port = htons(PORT); // short, network byte order
  printf("\n\n\nEnter IP address of the Server\n");
  scanf("%s", IP);
  server.sin_addr.s_addr = inet_addr(IP);
  memset(&(server.sin_zero), '\0', 8);

  if (connect(sockfd, (struct sockaddr *) &server, sizeof(struct sockaddr)) == -1) {
    perror("connect");
    exit(1);
  }

  if ((numbytes = recv(sockfd, buf, MAXDATASIZE - 1, 0)) == -1) {
    perror("recv");
    exit(1);
  }
  buf[numbytes] = '\0';
  printf("Received: %s", buf);

  close(sockfd);
  return 0;
}
```

C udpserver.c

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <stdlib.h>
int main() {
  int sock;
  int addr_len, bytes_read;
  char recv_data[1024];
  struct sockaddr_in server_addr, client_addr;
  if ((sock = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
    perror("Socket");
    exit(1);
  }
  server_addr.sin_family = AF_INET;
  server_addr.sin_port = htons(5000);
  server_addr.sin_addr.s_addr = INADDR_ANY;
  bzero(&(server_addr.sin_zero), 8);
  if (bind(sock, (struct sockaddr *) &server_addr, sizeof(struct sockaddr))
      == -1) {
    perror("Bind");
    exit(1);
  }
  addr_len = sizeof(struct sockaddr);
  printf("\nUDPServer Waiting for client on port 5000");
  fflush(stdout);
  while (1) {
    bytes_read = recvfrom(sock, recv_data, 1024, 0,
        (struct sockaddr *) &client_addr, &addr_len);
    recv_data[bytes_read] = '\0';
    printf("n(%s , %d) said : ", inet_ntoa(client_addr.sin_addr),
        ntohs(client_addr.sin_port));
    printf("%s", recv_data);
    fflush(stdout);
  }
  return 0;
}
```

D udpclient.c

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <stdio.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
int main() {
  int sock;
  struct sockaddr_in server_addr;
  struct hostent *host;
  char send_data[1024];
  host = (struct hostent *) gethostbyname((char *) "127.0.0.1");
  if ((sock = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
    perror("socket");
    exit(1);
  }
  server_addr.sin_family = AF_INET;
  server_addr.sin_port = htons(5000);
  server_addr.sin_addr = *((struct in_addr *) host->h_addr);
  bzero(&(server_addr.sin_zero), 8);
  while (1) {
    printf("Type Something (q or Q to quit):");
    gets(send_data);
    if ((strcmp(send_data, "q") == 0) || strcmp(send_data, "Q") == 0)
      break;
    else
      sendto(sock, send_data, strlen(send_data), 0,
          (struct sockaddr *) &server_addr, sizeof(struct sockaddr));
  }
}
```

References

[1] IETF Tools. Rfc793 - transmission control protocol. https://tools.ietf.org/html/rfc793, 1981.

[2] IETF Tools. Rfc768 - user datagram protocol. https://tools.ietf.org/html/rfc768, 1980.