Lab 1 – Socket Programming in C

CS3263 Embedded Computer Networks

Dept. of Computer Science and Engineering, University of Moratuwa

Learning Outcomes

After completing the lab, you will be able to

1. write a TCP client-server program and run
2. explain the functionalities of sockets

Introduction

In this lab we will use built-in C compiler in Ubuntu to compile socket programs that we will be writing in C.

1. Check whether gcc compiler is installed in your Ubuntu machine

```
gcc -v
```

2. If the output complains that gcc is unavailable, then install using the following commands

```
sudo apt update
sudo apt install build-essential sudo
apt-get install manpages-dev
```

verify again that gcc is inatalled using 'gcc -v'

3. You may write the following Hello World program in C to check whether the compiler works properly.

Open any text editor, type the following code, and save as test.c.

```
#include <stdio.h>
int main() {
      printf("Hello, world!\n");
      return 0;
}
```

Go to a terminal and go inside the folder in which you saved the above file.

```
gcc test.c
./a.out
```

You should see Hello World printed in the command line

4. For code writing we will install Visual Studio Code as the Integrated Development Environment (IDE)

```
sudo snap install --classic code
```

To open the IDE and create a file, just type 'code <filename>' in a terminal

Creating the Server

Go to a terminal and type the following command to start writing server code

```
code server.c
```

Include the relevant libraries by copying the following lines

```
#include<stdio.h>
#include<string.h>      //strlen
#include<sys/socket.h>
#include<arpa/inet.h>   //inet_addr
#include<unistd.h>      //write
```

Your whole program runs within the following main function.

```
int main(int argc , char *argv[])
{ return
0; }
```

Go inside the loop and fill the code as follows before 'return 0' line.

1. Define the variables

```
      int socket_desc , client_sock , c , read_size;
    struct sockaddr_in server , client;
char client_message[2000];
```

2. Create socket

A socket is an abstraction through which applications may send and receive date across networks. It uniquely identifies a process running in a computer across networks using an IP address (host identifier) and a port number (application identifier) as in Figure 1.

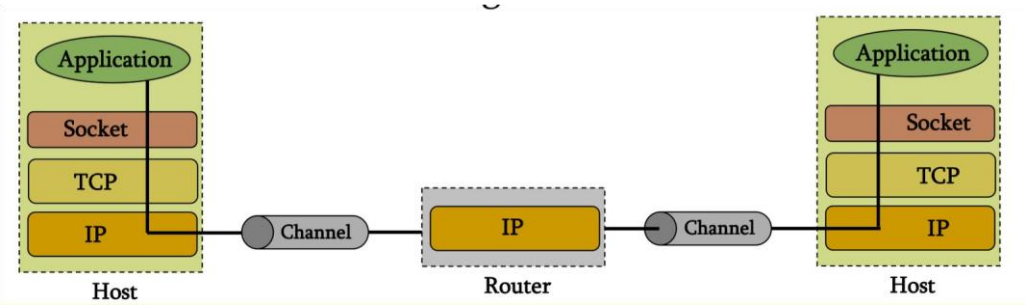

Figure 1: Socket as standard API for networking

//Create socket

```
socket_desc = socket(AF_INET , SOCK_STREAM , 0);
if (socket_desc == -1)
{ printf("Could not create socket");
} puts("Socket
created");
```

3. There are two types of sockets (see Figure 2), namely Stream sockets and Datagram sockets. In this lab exercise we use Stream sockets. We created a socket using socket() function in the previous step. In this step we will attach an IP address and a port and this process is called bind().

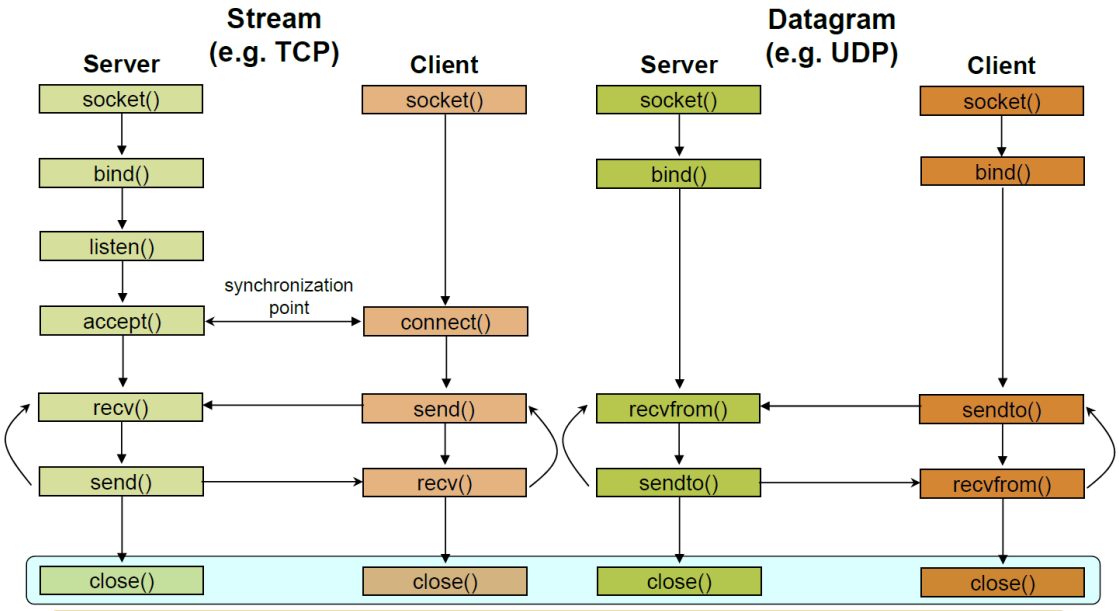

Figure 2

4. Bind() requires a struct of type sockaddr

```
     //Prepare the sockaddr_in structure
server.sin_family = AF_INET;
server.sin_addr.s_addr = INADDR_ANY;
server.sin_port = htons( 8888 );

     //Bind
     if( bind(socket_desc,(struct sockaddr *)&server , sizeof(server)) < 0)
     {
        //print the error message
perror("bind failed. Error");
        return 1;
     }
     puts("bind done");
```

5. Make the server announce the willingness to accept connections using listen() command

```
      //Listen       listen(socket_desc , 3);
puts("Waiting for incoming connections...");
```

6. Make the server accept incoming connection requests

```
      //Accept an incoming connection
c = sizeof(struct sockaddr_in);
       client_sock = accept(socket_desc,(struct sockaddr *)&client,
(socklen_t*)&c);

        if (client_sock < 0)
```

```
        {
                perror("accept failed");
return 1;
        }
        puts("Connection accepted");
```

7. Receive a message from client

```
      //Receive a message from client       while( (read_size =
recv(client_sock , client_message , 2000 , 0)) > 0 )
        {
                //Send the message back to client
                write(client_sock , client_message ,  strlen(client_message));
        }
      if(read_size == 0)
        {
                puts("Client disconnected");
fflush(stdout);
        }
        else if(read_size == -1)
        {
                perror("recv failed");
}
```

Creating the Client

1. Create a file called client.c
2. Include relevant libraries and write the main program

```
#include <stdio.h>       //printf
#include <string.h>     //strlen
#include <sys/socket.h> //socket
#include <arpa/inet.h>  //inet_addr
#include <unistd.h>

int main(int argc , char *argv[])
{   return 0;
}
```

3. Create a socket

```
      int sock;
       struct sockaddr_in server;
       char message[1000] , server_reply[2000];

       //Create socket
       sock = socket(AF_INET , SOCK_STREAM , 0);
if (sock == -1)
        {
                printf("Could not create socket");
        }
        puts("Socket created");
```

4. Connect to the server.

```
        server.sin_addr.s_addr = inet_addr("127.0.0.1");
        server.sin_family = AF_INET;
server.sin_port = htons( 8888 );

        //Connect to remote server
        if (connect(sock , (struct sockaddr *)&server , sizeof(server)) < 0)
        {
                perror("connect failed. Error");
return 1;
        }

        puts("Connected\n");
```

5. Send and receive data

```
        //keep communicating with server
while(1)
        {
                printf("Enter message : ");
scanf("%s" , message);

                //Send some data
                if( send(sock , message , strlen(message) , 0) < 0)
                {
                        puts("Send failed");
                        return 1;
                }
                //Receive a reply from the server
                if( recv(sock , server_reply , 2000 , 0) < 0)
                {
                        puts("recv failed");
break;
                }

                puts("Server reply :");
puts(server_reply);         }
```

6. Close the socket

```
close(sock);
```

Exercise

1. Upload your server.c and client.c files as txt files in Moodle.
2. Upload a pdf with screenshots of terminals when server and client are running. Please number the figures and give brief explanation about the figures by referring to the figure number.
3. Start capturing TCP only packets in Wireshark from localhost interface while the server is running. Run the client and explain the TCP packet trace. Please refer to message format and state diagram sections in RFC793. Explain observations in terms of flags, sequence number, acknowledgment number, source port, and destination port.

References

[1]. P. Fatourou, “Introduction to Socket Programming in C using TCP/IP,” online: https://www.csd.uoc.gr/~hy556/material/tutorials/cs556-3rd-tutorial.pdf, May 2012