|  |  |  |
| --- | --- | --- |
|  | **Namal Institute Mianwali** |  |
| Computer Networks Laboratory Manual #6 |
|  |
| **Network Programming using UNIX TCP/IP Sockets in C**  **Part-1: Echo Server and Client** | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Course Title | Computer Networks | Course Number | CS – 270 L |
| Instructor | Shahzad Arif  shahzad.arif@namal.edu.pk | Lab Engineer | Asad Majeed  [asad.majeed@namal.edu.pk](mailto:asad.majeed@namal.edu.pk) |

|  |  |
| --- | --- |
| Name | Muhammad Abrar Hussain |
| Roll No. | BSCS\_2020\_62 |

# 1. Introduction:

What should we do if we want to write our own application? In this Lab, we shall develop our knowledge of the Application Programming Interface (API) used to interact with the Operating System, to pass messages over the network using SOCKETS in C language.

Higher level languages such as Python, Java, .NET etc wrap these functions and use them. So, an understanding of the functions helps understand the underlying behaviour of the system.

# 2. Objective:

* To learn how to use Sockets API in C, for a UNIX TCP/IP network.
* To comprehend the implementation issues arising from buffer management.
* To be able to describe an Echo Server.

# 3. Methodology:

To achieve the above objectives, we shall use the development environment typically available on a Linux machine.

Fill in the Table (1) before starting to develop a code or a project, because the development environment changes with every upgrade of the development tools, operating system kernel and associated libraries.

Note: It is always a good practice to document the development environment, so that you can compile the code in future with known dependencies.

Table 1: Development Environment Details

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. no.** | **Tool / System / Library** | **Name** | **Version** | **Comments / Command to write in Terminal** |
| 1. | Linux Kernel | Linux | 5.13.0-41-generic | uname -r |
| 2. | C Compiler and Linker. | GNU C / C++ | 9.4.0 | gcc -v |
| 3. | System C Library | GNU glibc | Ldd (Ubuntu GLIBC 2.31-0ubuntu9.7) 2.31 | ldd --version |
| 4. | C debugger | GNU GDB | GNU gdb (Ubuntu 9.1-0ubuntu1) 9.1 | gdb -v |

# 4. Background:

Sockets allow you to exchange information between processes on the same machine or across a network. A wide range of operating systems support socket APIs. Sockets assist in programmers to implement **a client** and **a server**. A separate code is written for client, and server.

We shall establish a connection first before passing the messages, which is called “connection-oriented link”. In establishing a connection Transport Control Protocol (TCP) is required. Therefore, server characterised as a passive listener, is waiting for a connection request from a client.

Table 2: C Functions used in basic Socket API

|  |  |  |
| --- | --- | --- |
|  | **C Function** | **Manual** |
| Socket | int socket(int domain, int type, int protocol); | Man 2 socket |
| Connect | int connect(int sockfd, const struct sockaddr \*addr, socklen\_t addrlen); | Man 2 connect |
| Bind | int bind(int sockfd, const struct sockaddr \*addr, socklen\_t addrlen); | Man 2 bind |
| Listen | int listen(int sockfd, int backlog); | Man 2 listen |
| Accept | int accept(int sockfd, struct sockaddr \*addr, socklen\_t \*addrlen); | Man 2 accept |
| Write | ssize\_t write(int fd, const void \*buf, size\_t count); | Man 2 write |
| Read | ssize\_t read(int fd, void \*buf, size\_t count); | Man 2 read |
| Close | int close(int fd); | Man 2 close |
|  |  |  |

**Write down the description of each given function after reading from the manual.**

**Observation:**

**Socket:**

Socket () creates an endpoint for communication and returns a descriptor that refers to that endpoint. The file descriptor returned by a successful call will be the lowest numbered file descriptor not currently open for the process.

**Connect:**

The connect () system call connects the socket referred to by the file descriptor sockfd to the address specified by the address.

**Bind:**

When the socket is created it has no address. Bind assigns the address to the socket specified by the address referred to by the file descriptor sockfd.

**Listen:**

Listen to use to listen to the incoming requests on that IP and port. A server has a listen () method which puts the server into a mode

**Accept:**

Accept block execution and wait for an incoming connection. When a client connects, it returns a new socket object representing the connection and a tuple holding the address of the client.

**Write:**

Write () writes up to count bytes from the buffer starting at buf to the file referred to by the file descriptor fd.

**Read:**

Read (), read the bytes of count from the file descriptor fd into the buffer starting at buf.

**Close:**

Close () closes a file descriptor so that it no longer refers to any file and may be reused.

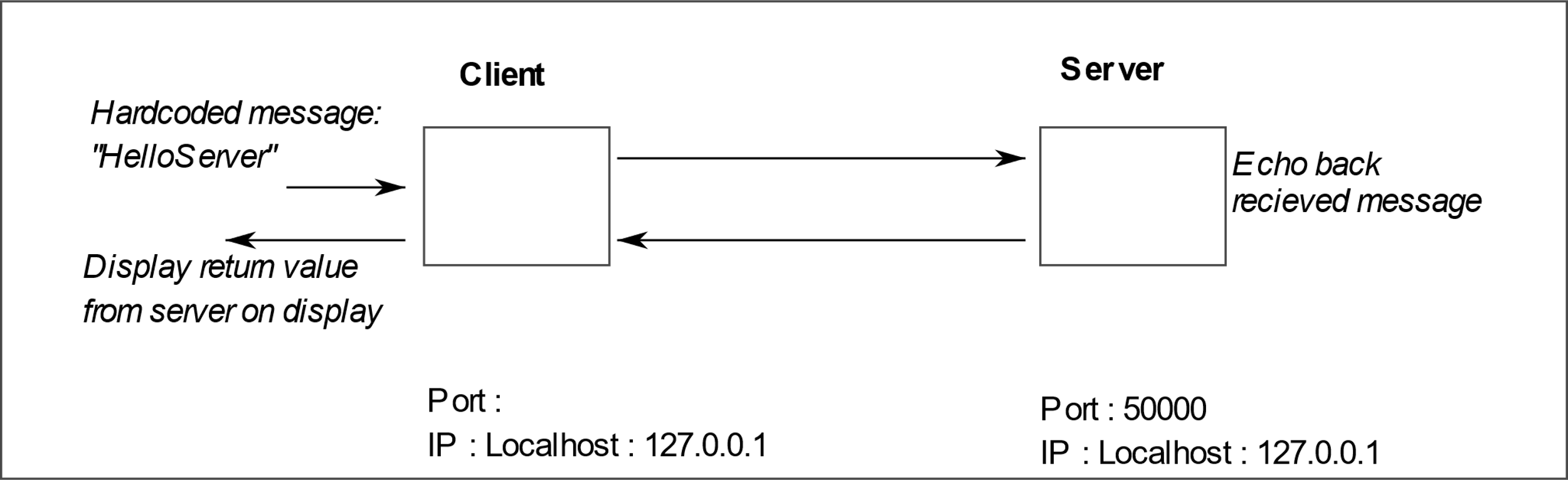
In the client-server model, the client should know the hostname and the port number on which the server is listening, while trying to fix the port number for incoming acknowledgements from server. These ports remain fixed for the duration of communication. If the server accepts the connection request, the server keep a note of the assigned ports, both at the client side as well as the server side. The server needs a new socket so that it can continue to listen to the original socket for connection requests while tending to the needs of the connected client. On the client side, if the connection is accepted, a socket is successfully created, and the client can use the socket to communicate with the server.

The client and server can now communicate by writing to or reading from their sockets.

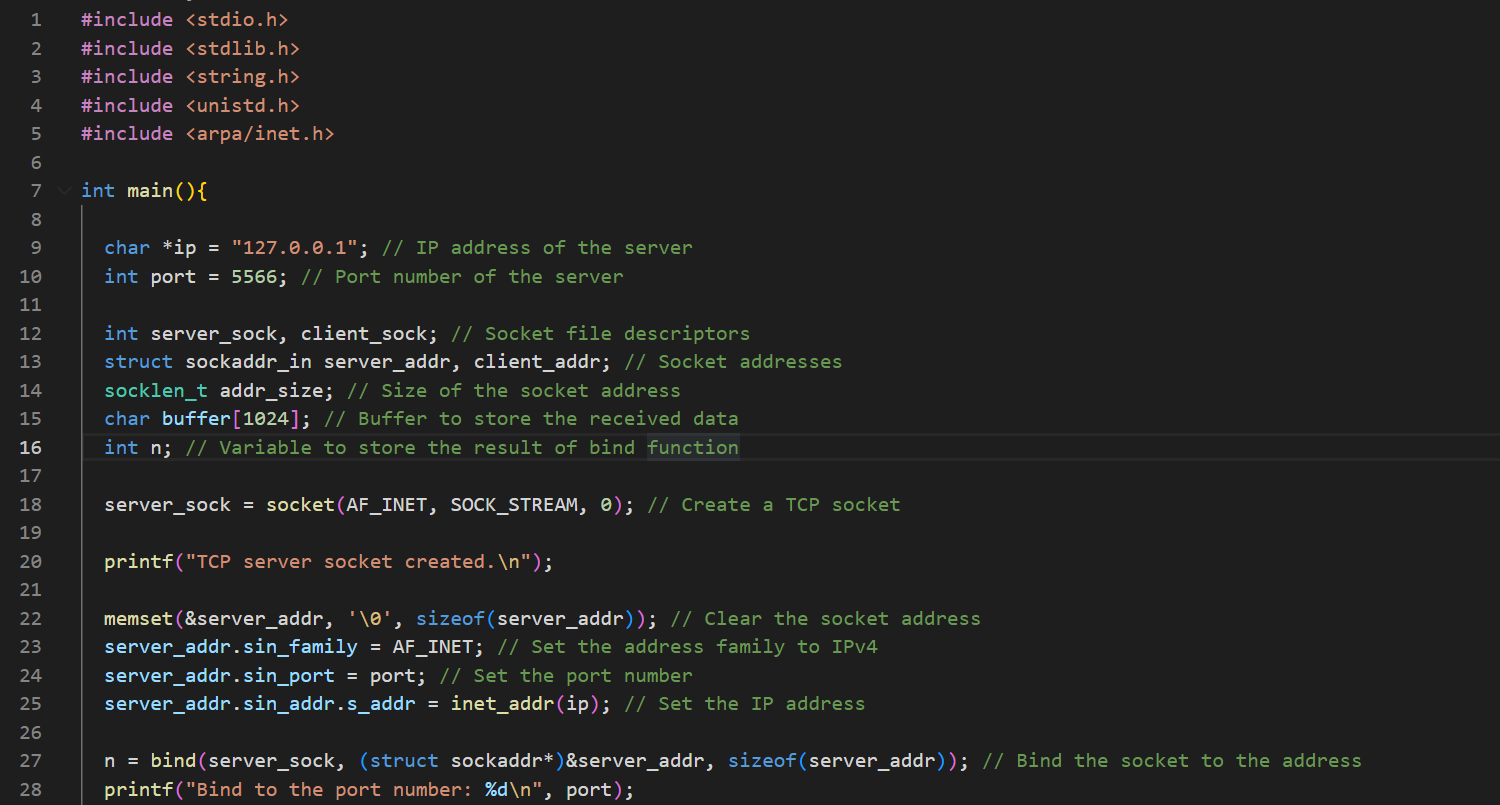
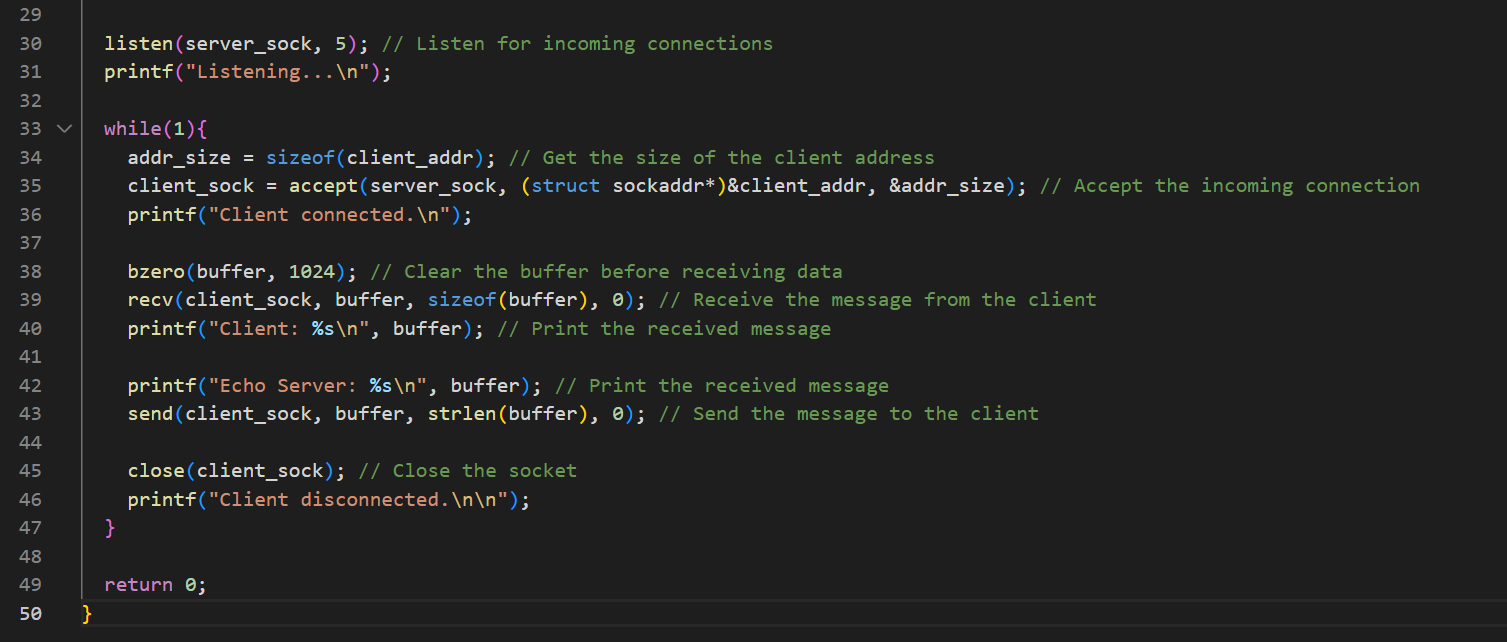
# 5. Scenario:

To write a complete TCP client/server, that performs the following steps:

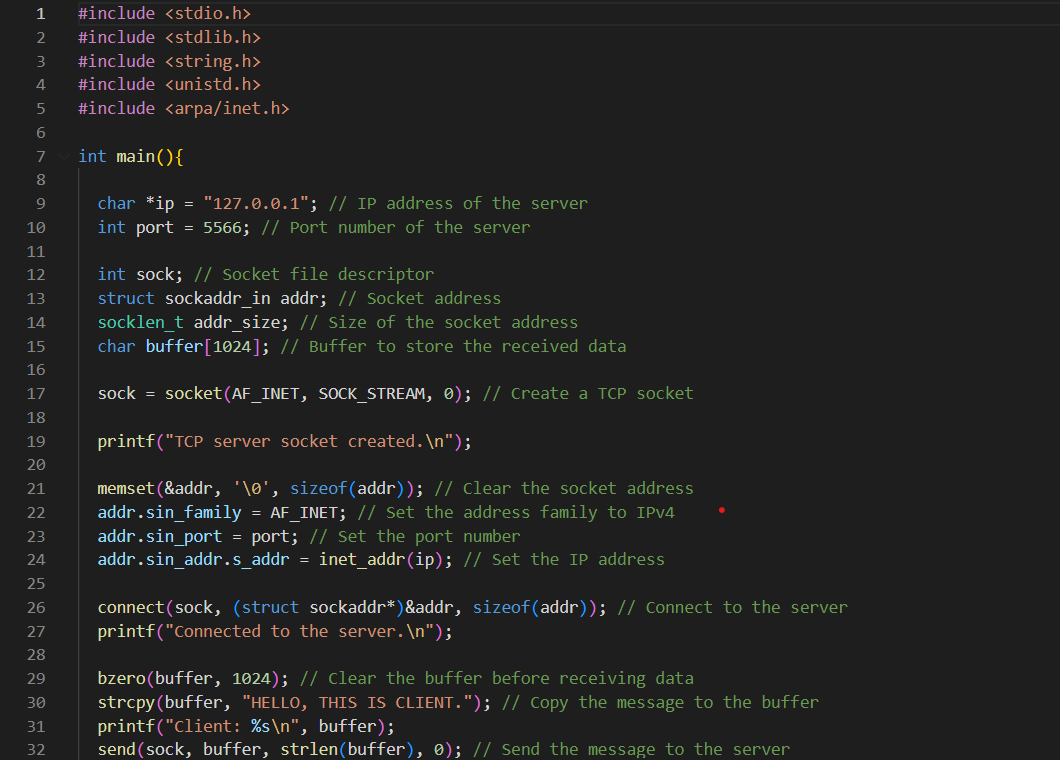
1. The client sends a hardcoded message “HelloServer” to the server.
2. The server reads the message from its network input and echoes the line back to the client.
3. The client reads the echoed line and prints it on its standard output i.e., the display.

Figure 2: Echo server-client scenario

## 5.1. Echo Server:



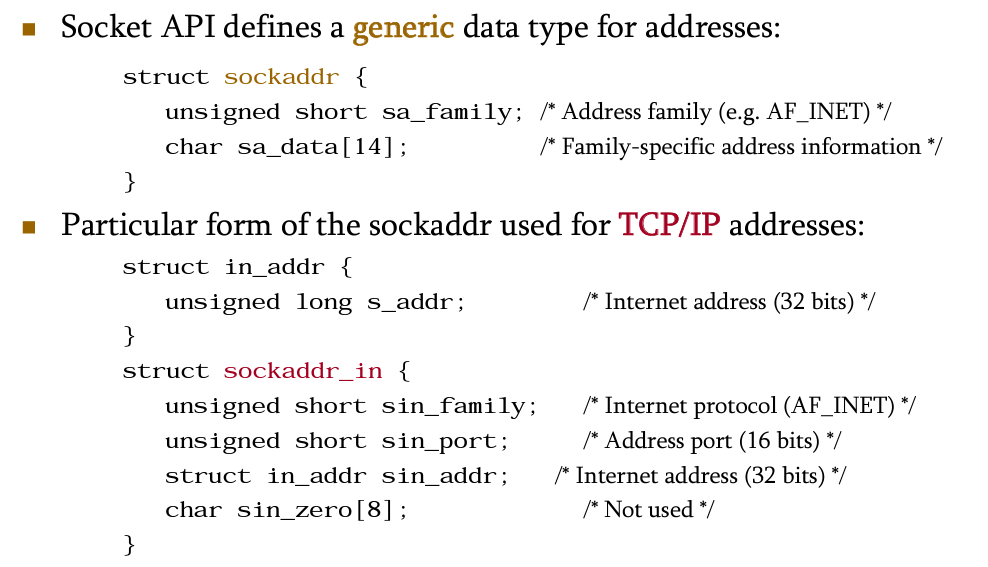
## 5.2 Echo Client:



Text

Description automatically generated

**Reference:**

****

## 5.3 Tasks

1. Write down the observed behavior of the given codes.
2. Modify both codes to check for errors. If any function returns -1, exit the main code.
3. Modify the Client Program to send a hardcoded message “Hello World” to the server and stop.

**End of Tasks**