Network Programming

Introduction to Sockets

Dr. Pradeep K V
School of Computing Science and Engineering
Vellore Institute of Technology
Chennai

Introduction

- Introduction to Socket
- Network Address
- Client/Server Model
- Structures
- Port and Services
- Network Byte Address
- IP Address Functions
- Socket System Calls
- Client/Server Example

Socket Means...

- Sockets are communication points on the same or different computers to exchange data.
- Sockets are supported by Unix, Windows, Mac, and many other operating systems.
- Sockets were first introduced in 2.1BSD and subsequently refined into their current form with 4.2BSD.
- A Socket is used in a client-server application framework.
- A server is a process that performs some functions on request from a client. Most of the application-level protocols like FTP, SMTP, and POP3 make use of sockets to establish connection between client and server and then for exchanging data.

Prerequisites :

- Learning Sockets is not at all a difficult task.
- You must be well versed with the basic concepts of C programming.

Types of Sockets

There are four types of sockets available to the users (First Two were used in common.)

- **Stream Sockets** : Guarantee Delivery (Arrive Packets in the same order), Uses TCP [Connection Oriented]. item If not Delivered, then sender received Error Indicator (NACK).
- **Datagram Sockets** : No-Guarantee Delivery, Uses UDP [ConnectionLess], No NACK.
- **Raw Sockets**: Used to develop new communication protocols, Datagram Oriented.
- **Sequenced Packet Sockets** : Similar to Stream Sockets
  - allow the user to manipulate the Sequence Packet Protocol (SPP) or Internet Datagram Protocol (IDP) headers on a packet or a group of packets

Network Address

- The IP host address is used to identify hosts connected to the Internet.
- IP stands for Internet Protocol and refers to the Internet Layer of the overall network architecture of the Internet.
- An IP address is a 32-bit quantity interpreted as four 8-bit numbers or octets. Each IP address uniquely identifies the participating user network, the host on the network, and the class of the user network.
- An IP address is usually written in a dotted-decimal notation of the form $N_1.N_2.N_3.N_4$, where each N_i is a decimal number between 0 and 255 decimal (00 through FF hexadecimal).

Socket Structures - **sockaddr**

- Various structures are used in Socket Programming to hold information about the **address** and **port**, and **other information**.
- Most socket functions require a pointer to a socket address structure as an argument.
- The first generic structure is **sockaddr** that holds the socket information

```
struct sockaddr
{
  unsigned short    sa_family;
  char              sa_data[14];
};
```

  - sa_family : an address family (AF_INET, AF_UNIX, AF_NS, AF_IMPLINK)
  - sa_data : The content of the 14 bytes of protocol specific address are interpreted according to the type of address.
    For the Internet family, we will use port number IP address, which is represented by **sockaddr_in** structure defined below.

Socket Structure - **sockaddr_in**

- **sockaddr_in** : is the second structure and as follows :

```
struct sockaddr_in
{
  short int          sin_family;
  unsigned short int sin_port;
  struct in_addr     sin_addr;
  unsigned char      sin_zero[8];
};
```

  - **sin_family** : represents an address family (AF_INET)
  - **sin_port** : A 16-bit port number in Network Byte Order.
  - **sin_addr** : A 32-bit IP address in Network Byte Order.
  - **sin_zero** : Not Used (Set as NULL)

- **in_addr** : structure is used to holds 32 bit **netid/hostid**.

```
struct in_addr {
  unsigned long s_addr;
   //A 32-bit IP address in Network Byte Order.
};
```

Socket Structure - **hostent**

- **hostent** : is used to keep information related to host.

```
struct hostent {
  char *h_name;
  char **h_aliases;
  int h_addrtype;
  int h_length;
  char **h_addr_list;

  #define h_addr  h_addr_list[0]
};
```

  - **h_name :** is the official name of the hos (Google.com)
  - **h_aliases :** It holds a list of host name aliases.
  - **h_addrtype :** It contains the address family and in case of Internet based application (AT_INET)
  - **h_length :** It holds the length of the IP address, which is 4 for Internet Address.
  - **h_addr_list :** For Internet addresses.

NOTE : **h_addr** is defined as **h_addr_list[0]** to keep backward compatibility.

Socket Structure : **servent**

- This particular structure is used to keep information related to service and associated ports.

```
struct servent {
    char  *s_name;
    char  **s_aliases;
    int   s_port;
    char  *s_proto;
};
```

  - **s_proto** : is the official name of the service.
    Eg : SMTP, FTP POP3, etc.
  - **s_aliases** : It holds the list of service aliases.(set to NULL).
  - **s_port** : It will have associated port number. Eg: HTTP - 80.
  - **s_proto** : It is set to the protocol used. (TCP/UDP)

Ports and Service

- When a client process wants to a connect a server, the client must have a way of identifying the server that it wants to connect.
- If the client knows the 32-bit Internet address of the host on which the server resides, it can contact that host.
- But how does the client identify the particular server process running on that host?
- To resolve the problem of identifying a particular server process running on a host, both TCP and UDP have defined a group of well-known ports.
- **PORT** : defined as an integer number between 1024 and 65535.
  - All port numbers smaller than 1024 are considered well-known ports

Examples of Port Numbers

Service	Port Number	Service Description
echo	7	UDP/TCP sends back what it receives.
discard	9	UDP/TCP throws away input.
daytime	13	UDP/TCP returns ASCII time.
chargen	19	UDP/TCP returns characters.
ftp	21	TCP file transfer.
telnet	23	TCP remote login.
smtp	25	TCP email.
daytime	37	UDP/TCP returns binary time.
tftp	69	UDP trivial file transfer.
finger	79	TCP info on users.
http	80	TCP World Wide Web.
login	513	TCP remote login.
who	513	UDP different info on users.
Xserver	6000	TCP X windows (N.B. >1023).

Network/Host Byte Address

- Ports and addresses are always to be specified well in socket
- All the socket functions uses the **Network byte order** convention(Big-Endian)
- This convention is a method of sorting bytes that is independent of specific machine architectures.
- **Host byte order**, on the other hand, sorts bytes in the manner which is most natural to the host software and hardware.
- Consider a 16-bit internet that is made up of 2 bytes. There are two ways to store this value.
  - **Little-endian** : In this scheme, low-order byte is stored on the starting address (A) and high-order byte is stored on the next address (A + 1). This method is used in Intel microprocessors.
  - **Big-endian** : In this schem, High-order byte is stored on the starting address (A) and low-order byte is stored on the next address (A + 1).. This method is used in IBM® z/Architecture® and S/390® mainframes and Motorola microprocessors

Byte Ordering Functions

Routines(Macros) for converting data between a Host's internal representation and Network Byte Order are as follows :

Function	**Description**
htons()	Host to Network Short
htonl()	Host to Network Long
ntohl()	Network to Host Long
ntohs()	Network to Host Short

- **unsigned short htons(unsigned short hostshort)** : It converts 16-bit (2-byte) quantities from host byte order to network byte order.
- **unsigned long htonl(unsigned long hostlong)** :It converts 32-bit (4-byte) quantities from host byte order to network byte order.
- **unsigned short ntohs(unsigned short netshort)** : It converts 16-bit (2-byte) quantities from network byte order to host byte order.
- **unsigned long ntohl(unsigned long netlong)** : It converts 32-bit quantities from network byte order to host byte order.

IP Address Functions

The following three function calls are used for IPv4 addressing :

- **int inet_aton(const char *strptr, struct in_addr *addrptr)**
- **in_addr_t inet_addr(const char *strptr)**
- **char *inet_ntoa(struct in_addr inaddr)**

int inet_aton() : converts the specified string in the Internet standard dot notation to a network address, and stores the address in the structure provided.

```
#include <arpa/inet.h>

   (...)

   int retval;
   struct in_addr addrptr

   memset(&addrptr, '\0', sizeof(addrptr));
   retval = inet_aton("68.178.157.132", &addrptr);

   (...)
```

IP Address Functions.....

in_addr_t inet_addr() : converts the specified string in the Internet standard dot notation to an integer value suitable for use as an Internet address. It returns a 32-bit binary network byte ordered IPv4 address and INADDR_NONE on error.

```
#include <arpa/inet.h>

   (...)

   struct sockaddr_in dest;

   memset(&dest, '\0', sizeof(dest));
   dest.sin_addr.s_addr = inet_addr("68.178.157.132");

   (...)
```

IP Address Functions

char *inet_ntoa() : converts the specified Internet host address to a string in the Internet standard dot notation.

```
#include <arpa/inet.h>

    (...)

    char *ip;

    ip = inet_ntoa(dest.sin_addr);

    printf("IP Address is: %s\n",ip);

    (...)
```

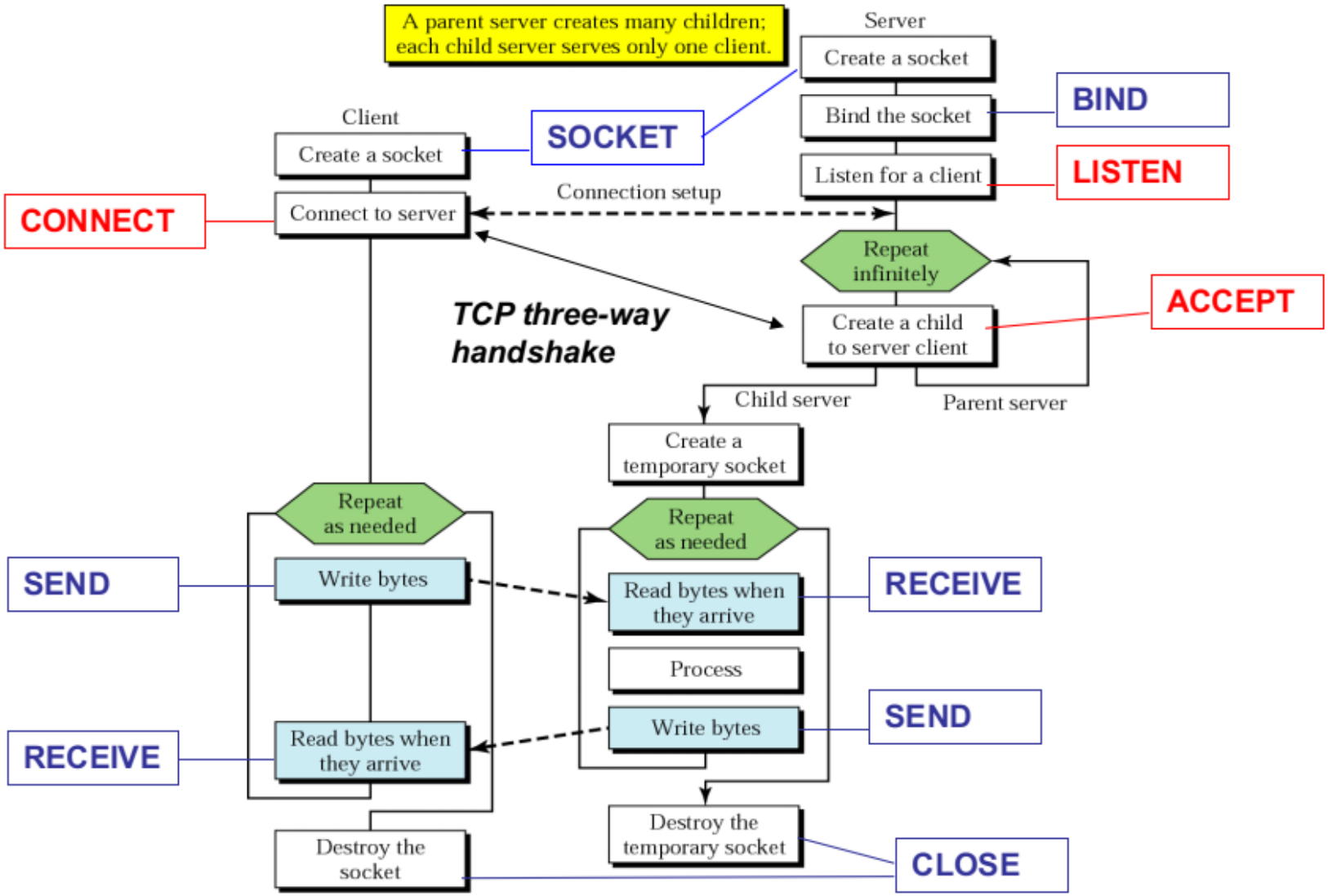
Client/Server Model

- Most of the N/w Applications use the Client-Server architecture, which refers to two processes or two applications that communicate with each other to exchange some information.
- One of the two processes acts as a client process, and another process acts as a server.
- **Client Process** : It sends a request for information to Server.
  Example : Internet Browser works as a client application, which sends a request to the Web Server to get one HTML webpage.
- **Server Process** : which takes a request from the clients, process it and responds back to client.
  Example : Web Server keeps waiting for requests from Internet Browsers and as soon as it gets any request from a browser, it picks up a requested HTML page and sends it back to that Browser.

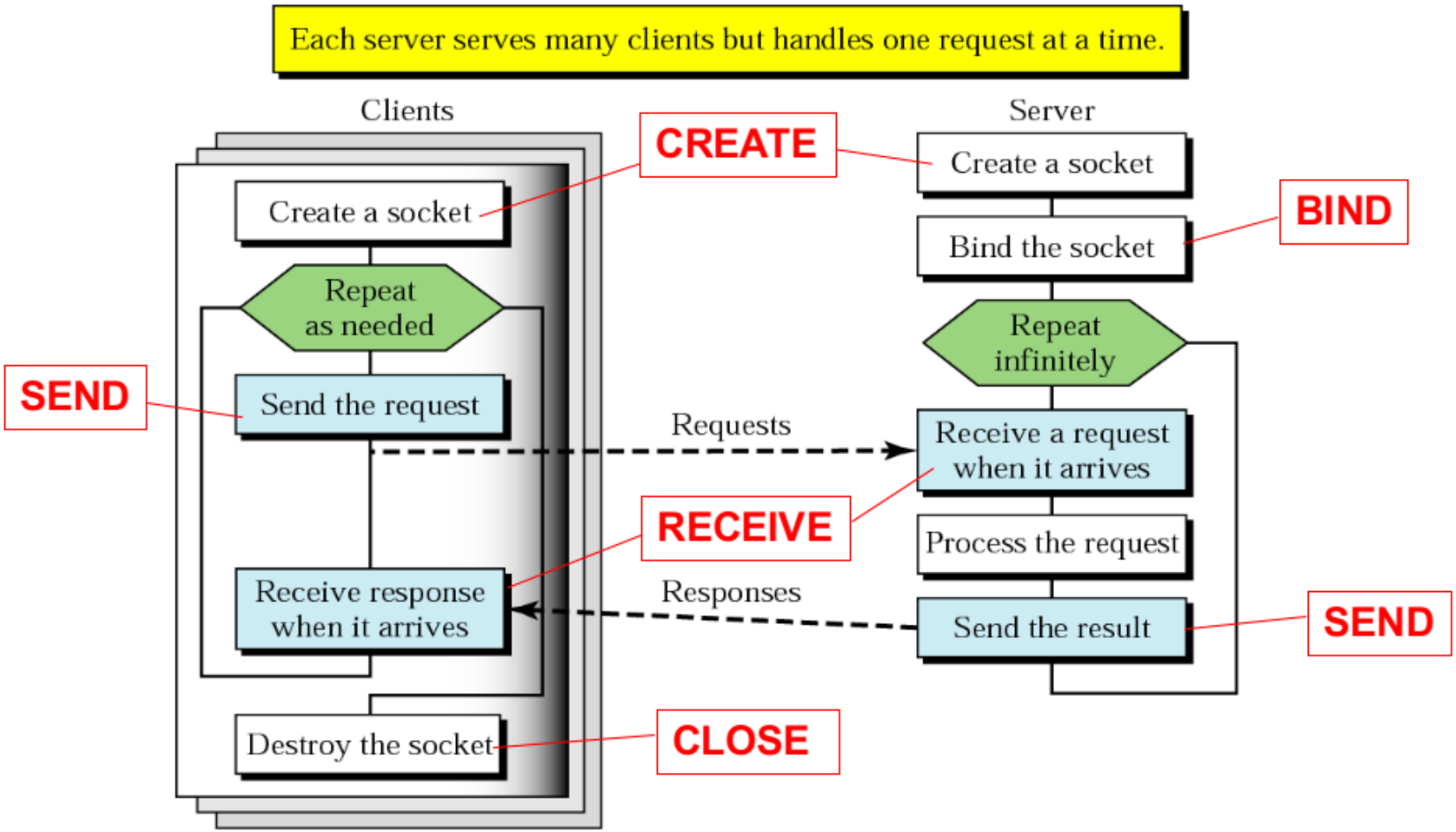
Types of Servers

- **Iterative Server** : This is the simplest form of server where a server process serves one client and after completing the first request, it takes request from another client. Meanwhile, another client keeps waiting.
- **Concurrent Server** : This type of server runs multiple concurrent processes to serve many requests at a time because one process may take longer and another client cannot wait for so long. The simplest way to write a concurrent server under Unix is to fork a child process to handle each client separately.

Client/Server Connection Oriented



Client/Server Connectionless



Socket System Calls

1. socket() : To Create Socket (TCP/UDP - Client/Server)
2. bind() : To Bind the Server Address to a Socket (TCP - Server)
3. connect() : To establish a connection with a Server (TCP Client).
4. listen() : used by TCP Server
5. accept() : used by TCP Server to accept Conncetion
6. send() : To Send messages over TCP Connection
7. recv() : To Receive messages over TCP Connection
8. sendto() : To Send messages over UDP Connection
9. recvfrom() : To Receive messages over UDP Connection
10. write() : To Write messages on to a TCP Socket :
11. read() : To Receive messages from TCP Socket
12. close() : Close the Connection (TCP/UDP)

Socket System Calls - **socket()**

- socket() : To perform network I/O, the first thing a process must do is, call the socket function, specifying the type of communication protocol desired and protocol family, etc.

```
#include <sys/types.h>
#include <sys/socket.h>

int socket (int family, int type, int protocol);
```

Returns : a socket descriptor that you can use in later system calls or -1 on error.

family : It specifies the protocol family and is one of the constants shown below

Family	**Description**
AF_INET	IPv4 protocols
AF_INET6	IPv6 protocols
AF_LOCAL	Unix domain protocols
AF_ROUTE	Routing Sockets
AF_KEY	Ket socket

Socket System Calls - **socket()**

type : It specifies the kind of socket you want. It can take one of the following values

Type	Description
SOCK_STREAM	Stream socket
SOCK_DGRAM	Datagram socket
SOCK_SEQPACKET	Sequenced packet socket
SOCK_RAW	Raw socket

protocol : The argument should be set to the specific protocol type given below, or 0 to select the system's default for the given combination of family and type :

Protocol	Description
IPPROTO_TCP	TCP transport protocol
IPPROTO_UDP	UDP transport protocol
IPPROTO_SCTP	SCTP transport protocol

Socket System Calls - **connect()**

- **connect()** : is used by a TCP client to establish a connection with a TCP server.

```
#include <sys/types.h>
#include <sys/socket.h>

int connect(int sockfd, struct sockaddr *serv_addr,
  int addrlen);
```

Returns : 0 if it successfully connects to the server, otherwise it returns -1 on error.

- parameters :
  - **sockfd** : It is a socket descriptor returned by the socket function.
  - **serv_addr** : It is a pointer to struct sockaddr that contains destination IP address and port.
  - **addrlen** : Set it to sizeof(struct sockaddr).

Socket System call : **bind()**

- **bind()** : It assigns a local protocol address to a socket. The protocol address is the combination of either a 32-bit IPv4 address or a 128-bit IPv6 address, along with a 16-bit TCP or UDP port number.

```
#include <sys/types.h>
#include <sys/socket.h>

int bind(int sockfd, struct sockaddr *my_addr,int
addrlen);
```

  Returns : 0 if it successfully binds to the address, otherwise it returns -1 on error.

- Parameters :
  - **sockfd** : It is a socket descriptor returned by the socket function.
  - **my_addr** : It is a pointer to struct sockaddr that contains the local IP address and port.
  - **addrlen** : Set it to sizeof(struct sockaddr).

A 0 value for port number means that the system will choose a random port, and INADDR_ANY value for IP address means the server's IP address will be assigned automatically.

Socket System Call - **listen()**

- **listen()** : it performs two actions
  - It converts an unconnected socket into a passive socket, indicating that the kernel should accept incoming connection requests directed to this socket.
  - The second argument to this function specifies the maximum number of connections the kernel should queue for this socket.

```
#include <sys/types.h>
#include <sys/socket.h>

int listen(int sockfd,int backlog);
```

Returns : 0 on success, otherwise it returns -1 on error.

  - parameters :
    - **sockfd** : It is a socket descriptor returned by the socket function.
    - **backlog** : It is the number of allowed connections.

Socket System Call - **accept()**

- **accept()** : used to accept the connection from Client.

```
#include <sys/types.h>
#include <sys/socket.h>

int accept (int sockfd, struct sockaddr *cliaddr,
socklen_t *addrlen);
```

Returns : 0 on success, otherwise it returns -1 on error.

- **parameters** :
  - **sockfd** : It is a socket descriptor returned by the socket function.
  - **cliaddr** : It is a pointer to struct sockaddr that contains client IP address and port.
  - **addrlen** : Set it to sizeof(struct sockaddr).

Socket System Call - **send()/recv()**

- **send()/write()** : is used to send data over stream sockets
- **recv()/read()** : is used to receive data over stream sockets

```
#include <sys/types.h>
#include <sys/socket.h>

int send(int sockfd, const void *msg, int len, int
  flags);

int recv(int sockfd, void *buf, int len, unsigned int
  flags);
```

Returns : number of bytes sent out /read in, otherwise it will return -1 on error.

- **parameters** :
  - **sockfd** : It is a socket descriptor returned by the socket function.
  - **msg** : It is a pointer to the data you want to send.
  - **buf** : It is the buffer to read the information into.
  - **len** : It is the length of the data you want to send/recv (in bytes).
  - **flags** : It is set to 0.

Socket System Call - **sendto()/recvfrom()**

- **sendto()** : is used to send data over datagram sockets
- **recvfrom()** : is used to receive data over datagram sockets

```
#include <sys/types.h>
#include <sys/socket.h>

int sendto(int sockfd, const void *msg, int len,
unsigned int flags, const struct sockaddr *to, int
tolen);

int recvfrom(int sockfd, void *buf, int len,
unsigned int flags struct sockaddr *from, int *
fromlen);
```

Returns : number of bytes on sucess, otherwise -1 on error.

- **parameters** :
  - **sockfd** : It is a socket descriptor returned by the socket function.
  - **msg** : It is a pointer to the data you want to send.
  - **buf** : It is the buffer to read the information into.
  - **to/from** : a pointer to struct sockaddr for the host where data has to be sent/recv.
  - **tolen/fromlen** : It is set it to sizeof(struct sockaddr).

Socket System Call - **close()/shutdown()**

- **close()/shutdown()** : is used to close the communication between the client and the server.

```
#include <sys/types.h>
#include <sys/socket.h>

int close( int sockfd );
int shutdown(int sockfd, int how);
```

Returns : 0 on sucess, otherwise -1 on error.

- **parameters** :
  - **sockfd** : is a socket descriptor returned by the socket function.
  - **how** :
    - 0 : indicates that receiving is not allowed,
    - 1 : indicates that sending is not allowed, and
    - 2 : indicates that both sending and receiving are not allowed. When how is set to 2, it's the same thing as close().

Example : TCP Server

```
#include "kvp.h"

int main(int argc, char *argv[])
{
  int listenfd = 0, connfd = 0;
  struct sockaddr_in serv_addr;

  char sendBuff[1025];
  time_t ticks;

  listenfd = socket(AF_INET, SOCK_STREAM, 0);
  memset(&serv_addr, '0', sizeof(serv_addr));
  memset(sendBuff, '0', sizeof(sendBuff));

  serv_addr.sin_family = AF_INET;
  serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
  serv_addr.sin_port = htons(5000);

  bind(listenfd, (struct sockaddr*)&serv_addr,
    sizeof(serv_addr));

   listen(listenfd, 10);
```

Example : TCP Server....

```
    bind(listenfd, (struct sockaddr*)&serv_addr, sizeof(
    serv_addr));

    listen(listenfd, 10);

    while(1)
    {
      connfd = accept(listenfd, (struct sockaddr*)NULL,
                NULL);

      ticks = time(NULL);

      snprintf(sendBuff, sizeof(sendBuff), "%.24s\r\n",
          ctime(&ticks));

      write(connfd, sendBuff, strlen(sendBuff));

      close(connfd);
      sleep(1);
    }
}
```

Example : TCP Client

```c
#include "kvp.h"

int main(int argc, char *argv[])
{
  int sockfd = 0, n = 0;
  char recvBuff[1024];
  struct sockaddr_in serv_addr;

  if(argc != 2)
  {
    printf("\n Usage: %s <ip of server> \n",argv[0]);
    return 1;
  }

  memset(recvBuff, '0',sizeof(recvBuff));

  if((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
  {
    printf("\n Error : Could not create socket \n");
    return 1;
  }

  memset(&serv_addr, '0', sizeof(serv_addr));
```

Example : TCP Client...

```
serv_addr.sin_family = AF_INET;
serv_addr.sin_port = htons(5000);

if(inet_pton(AF_INET, argv[1], &serv_addr.sin_addr)<=0)
{
   printf("\n inet_pton error occured\n");
   return 1;
}

if( connect(sockfd, (struct sockaddr *)&serv_addr,
sizeof(serv_addr)) < 0)
{
   printf("\n Error : Connect Failed \n");
   return 1;
}
while ( (n = read(sockfd, recvBuff, sizeof(recvBuff)-1))
          > 0)
{
   recvBuff[n] = 0;
   if(fputs(recvBuff, stdout) == EOF)
       printf("\n Error : Fputs error\n");
}
```

Example : TCP Client...

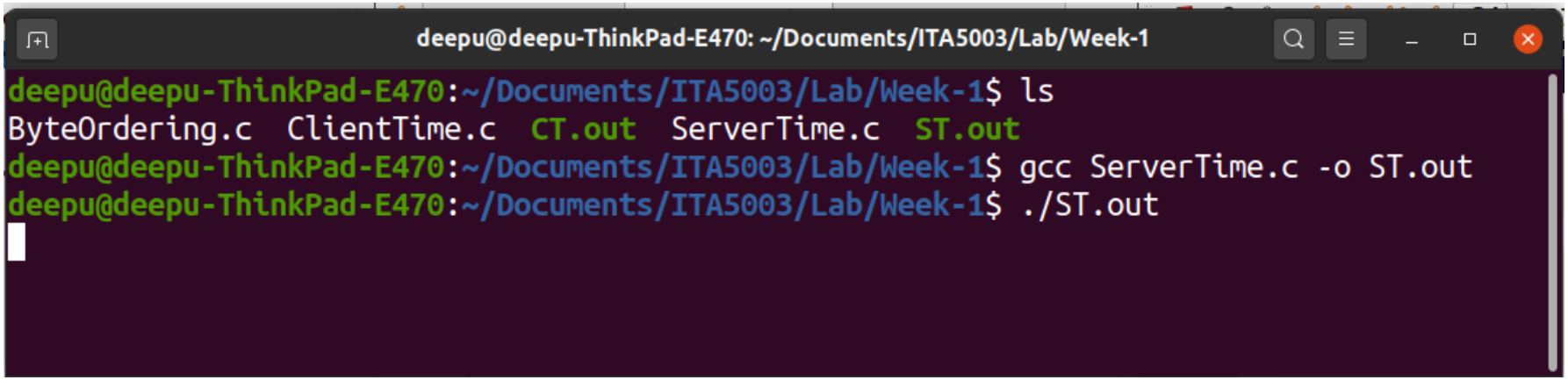
```
    if(n < 0)
    {
       printf("\n Read error \n");
    }

    return 0;
}
```

The Header File "kvp.h" contains

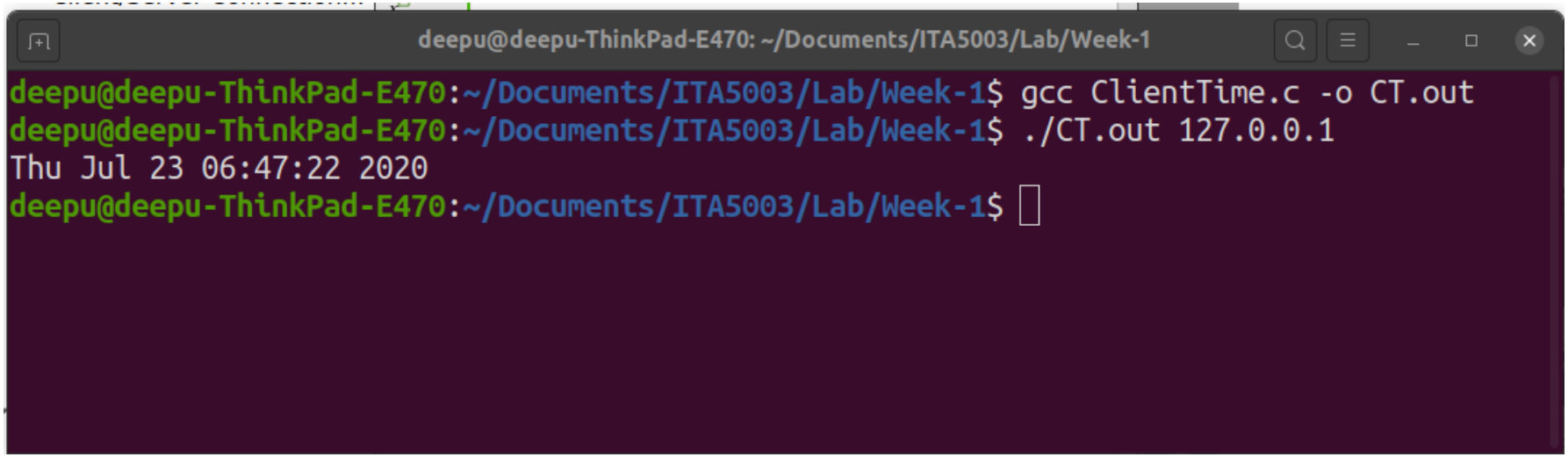
```
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <arpa/inet.h>
```

Output

TCP Server



TCP Client



Thanks...!