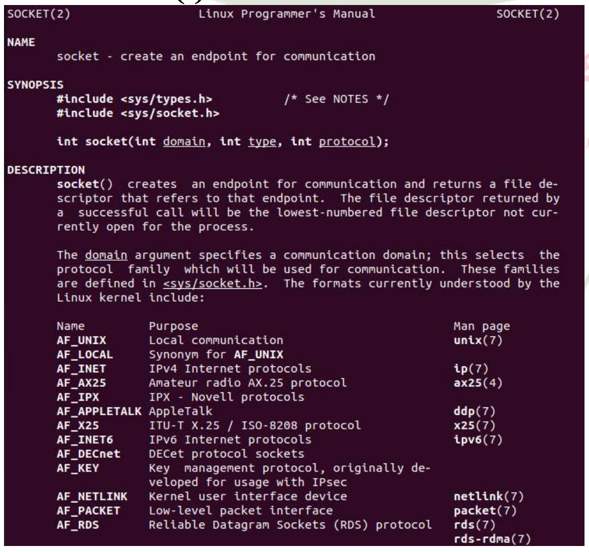
EXPERIMENT - 6

AIM: IMPLEMENATION OF SOCKET COMMANDS SUCH AS SOCKET, SEND, RECV, BIND, LISTEN, ACCEPT, CONNECT.

DESCRIPTION:

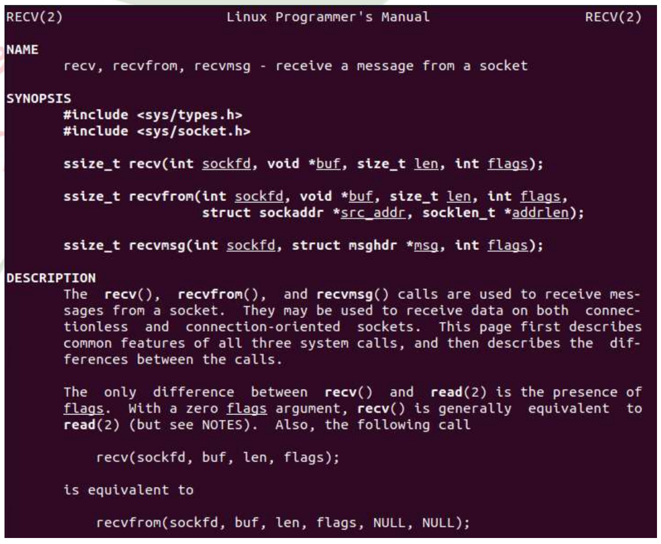
1. **socket()**: Initializes a new socket instance for network communication, specifying the address family and type (e.g., TCP/UDP).
2. **send()**: Sends data to a connected socket in a TCP connection, used for transmitting messages between client and server.
3. **recv()**: Receives data from a connected socket in a TCP connection, waiting for incoming data from the peer.
4. **bind()**: Assigns a specific local IP address and port to a socket, preparing it for listening to incoming connections.
5. **listen()**: Puts the socket in a passive mode, allowing it to accept incoming connection requests from clients.
6. **accept()**: Accepts an incoming connection on a listening socket, creating a new socket to handle client communication.
7. **connect()**: Initiates a connection to a remote server, allowing the client to establish a communication channel.

SOCKET( )



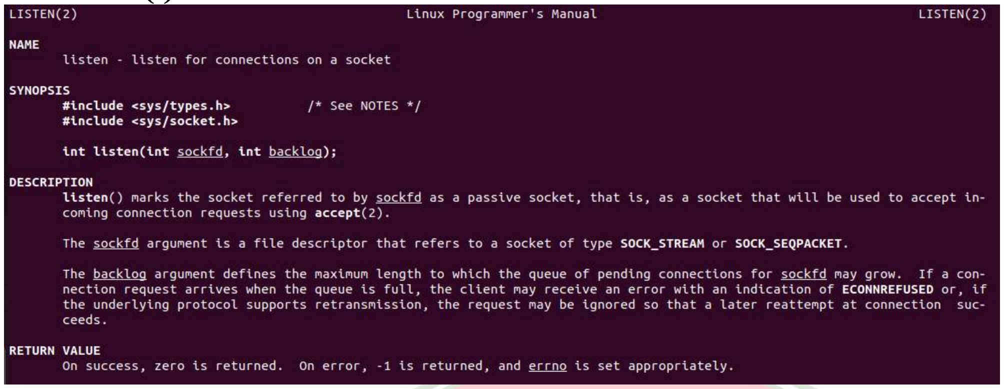
```
SOCKET(2)                  Linux Programmer's Manual                 SOCKET(2)

NAME
       socket - create an endpoint for communication

SYNOPSIS
       #include <sys/types.h>          /* See NOTES */
       #include <sys/socket.h>

       int socket(int domain, int type, int protocol);

DESCRIPTION
       socket()  creates  an endpoint for communication and returns a file de‐
       scriptor that refers to that endpoint.  The file descriptor returned by
       a  successful call will be the lowest-numbered file descriptor not cur‐
       rently open for the process.

       The domain argument specifies a communication domain; this selects  the
       protocol  family  which will be used for communication.  These families
       are defined in <sys/socket.h>.  The formats currently understood by the
       Linux kernel include:

       Name         Purpose                                    Man page
       AF_UNIX      Local communication                        unix(7)
       AF_LOCAL     Synonym for AF_UNIX
       AF_INET      IPv4 Internet protocols                    ip(7)
       AF_AX25      Amateur radio AX.25 protocol               ax25(4)
       AF_IPX       IPX - Novell protocols
       AF_APPLETALK AppleTalk                                  ddp(7)
       AF_X25       ITU-T X.25 / ISO-8208 protocol             x25(7)
       AF_INET6     IPv6 Internet protocols                    ipv6(7)
       AF_DECnet    DECet protocol sockets
       AF_KEY       Key  management protocol, originally de‐
                    veloped for usage with IPsec
       AF_NETLINK   Kernel user interface device               netlink(7)
       AF_PACKET    Low-level packet interface                 packet(7)
       AF_RDS       Reliable Datagram Sockets (RDS) protocol   rds(7)
                                                               rds-rdma(7)
```

RECV( )



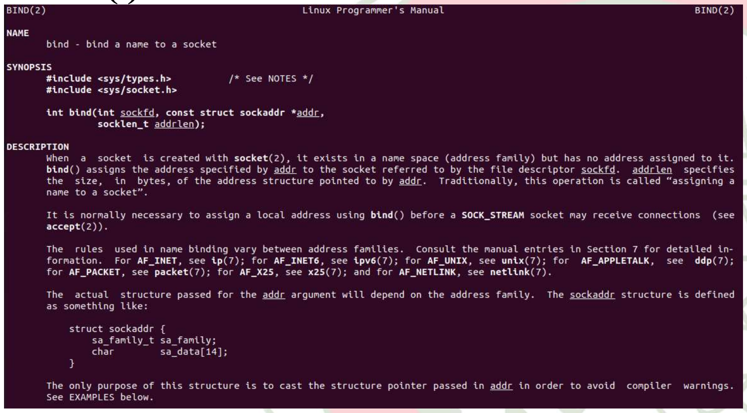
```
RECV(2)                    Linux Programmer's Manual                   RECV(2)

NAME
       recv, recvfrom, recvmsg - receive a message from a socket

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

       ssize_t recv(int sockfd, void *buf, size_t len, int flags);

       ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags,
                        struct sockaddr *src_addr, socklen_t *addrlen);

       ssize_t recvmsg(int sockfd, struct msghdr *msg, int flags);

DESCRIPTION
       The  recv(),  recvfrom(),  and recvmsg() calls are used to receive mes‐
       sages from a socket.  They may be used to receive data on both  connec‐
       tionless  and  connection-oriented  sockets.  This page first describes
       common features of all three system calls, and then describes the  dif‐
       ferences between the calls.

       The  only  difference  between  recv()  and  read(2) is the presence of
       flags.  With a zero flags argument, recv() is generally  equivalent  to
       read(2) (but see NOTES).  Also, the following call

           recv(sockfd, buf, len, flags);

       is equivalent to

           recvfrom(sockfd, buf, len, flags, NULL, NULL);
```

LISTEN( )



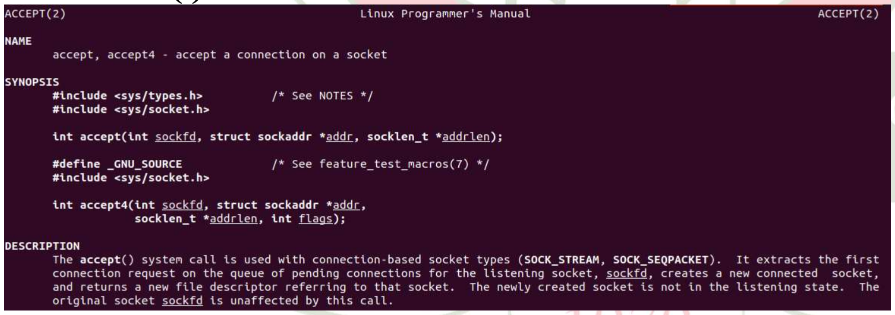
```
LISTEN(2)                        Linux Programmer's Manual                        LISTEN(2)

NAME
       listen - listen for connections on a socket

SYNOPSIS
       #include <sys/types.h>          /* See NOTES */
       #include <sys/socket.h>

       int listen(int sockfd, int backlog);

DESCRIPTION
       listen() marks the socket referred to by sockfd as a passive socket, that is, as a socket that will be used to accept in‐
       coming connection requests using accept(2).

       The sockfd argument is a file descriptor that refers to a socket of type SOCK_STREAM or SOCK_SEQPACKET.

       The backlog argument defines the maximum length to which the queue of pending connections for sockfd may grow.  If a con‐
       nection request arrives when the queue is full, the client may receive an error with an indication of ECONNREFUSED or, if
       the underlying protocol supports retransmission, the request may be ignored so that a later reattempt at connection  suc‐
       ceeds.

RETURN VALUE
       On success, zero is returned.  On error, -1 is returned, and errno is set appropriately.
```

BIND( )



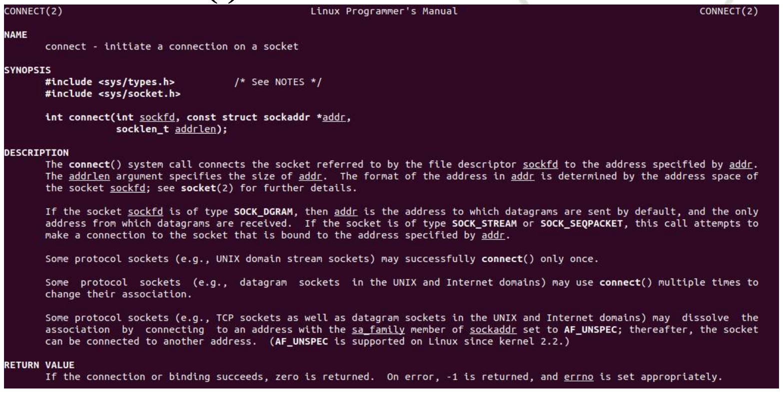
```
BIND(2)                          Linux Programmer's Manual                          BIND(2)

NAME
       bind - bind a name to a socket

SYNOPSIS
       #include <sys/types.h>          /* See NOTES */
       #include <sys/socket.h>

       int bind(int sockfd, const struct sockaddr *addr,
                socklen_t addrlen);

DESCRIPTION
       When  a  socket  is created with socket(2), it exists in a name space (address family) but has no address assigned to it.
       bind() assigns the address specified by addr to the socket referred to by the file descriptor sockfd.  addrlen  specifies
       the  size,  in  bytes, of the address structure pointed to by addr.  Traditionally, this operation is called “assigning a
       name to a socket”.

       It is normally necessary to assign a local address using bind() before a SOCK_STREAM socket may receive connections  (see
       accept(2)).

       The  rules  used in name binding vary between address families.  Consult the manual entries in Section 7 for detailed in‐
       formation.  For AF_INET, see ip(7); for AF_INET6, see ipv6(7); for AF_UNIX, see unix(7); for  AF_APPLETALK,  see  ddp(7);
       for AF_PACKET, see packet(7); for AF_X25, see x25(7); and for AF_NETLINK, see netlink(7).

       The  actual  structure passed for the addr argument will depend on the address family.  The sockaddr structure is defined
       as something like:

           struct sockaddr {
               sa_family_t sa_family;
               char        sa_data[14];
           }

       The only purpose of this structure is to cast the structure pointer passed in addr in order to avoid  compiler  warnings.
       See EXAMPLES below.
```

ACCEPT( )



```
ACCEPT(2)                        Linux Programmer's Manual                        ACCEPT(2)

NAME
       accept, accept4 - accept a connection on a socket

SYNOPSIS
       #include <sys/types.h>          /* See NOTES */
       #include <sys/socket.h>

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

       #define _GNU_SOURCE             /* See feature_test_macros(7) */
       #include <sys/socket.h>

       int accept4(int sockfd, struct sockaddr *addr,
                   socklen_t *addrlen, int flags);

DESCRIPTION
       The accept() system call is used with connection-based socket types (SOCK_STREAM, SOCK_SEQPACKET).  It extracts the first
       connection request on the queue of pending connections for the listening socket, sockfd, creates a new connected  socket,
       and returns a new file descriptor referring to that socket.  The newly created socket is not in the listening state.  The
       original socket sockfd is unaffected by this call.
```

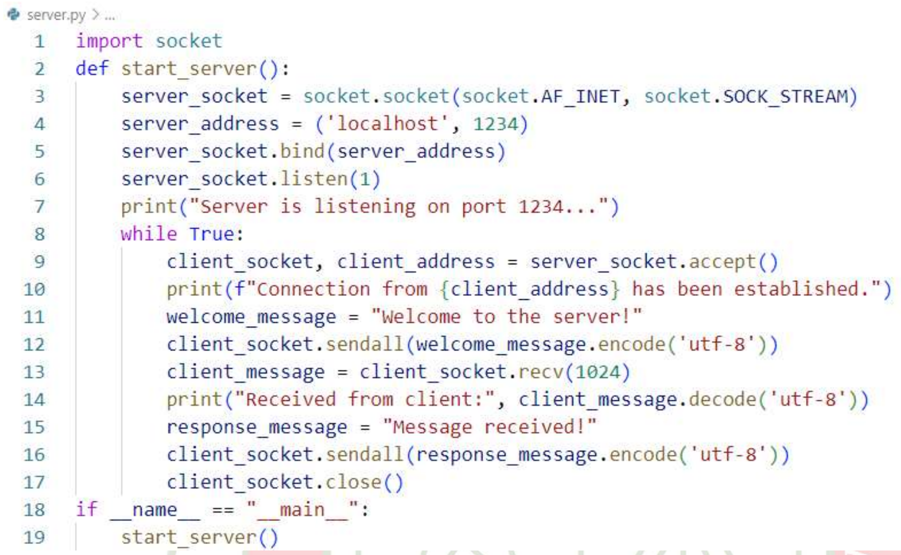
CONNECT( )



```
CONNECT(2)                       Linux Programmer's Manual                       CONNECT(2)

NAME
       connect - initiate a connection on a socket

SYNOPSIS
       #include <sys/types.h>          /* See NOTES */
       #include <sys/socket.h>

       int connect(int sockfd, const struct sockaddr *addr,
                   socklen_t addrlen);

DESCRIPTION
       The connect() system call connects the socket referred to by the file descriptor sockfd to the address specified by addr.
       The addrlen argument specifies the size of addr.  The format of the address in addr is determined by the address space of
       the socket sockfd; see socket(2) for further details.

       If the socket sockfd is of type SOCK_DGRAM, then addr is the address to which datagrams are sent by default, and the only
       address from which datagrams are received.  If the socket is of type SOCK_STREAM or SOCK_SEQPACKET, this call attempts to
       make a connection to the socket that is bound to the address specified by addr.

       Some protocol sockets (e.g., UNIX domain stream sockets) may successfully connect() only once.

       Some  protocol  sockets  (e.g.,  datagram  sockets  in the UNIX and Internet domains) may use connect() multiple times to
       change their association.

       Some protocol sockets (e.g., TCP sockets as well as datagram sockets in the UNIX and Internet domains) may  dissolve  the
       association  by  connecting  to an address with the sa_family member of sockaddr set to AF_UNSPEC; thereafter, the socket
       can be connected to another address.  (AF_UNSPEC is supported on Linux since kernel 2.2.)

RETURN VALUE
       If the connection or binding succeeds, zero is returned.  On error, -1 is returned, and errno is set appropriately.
```

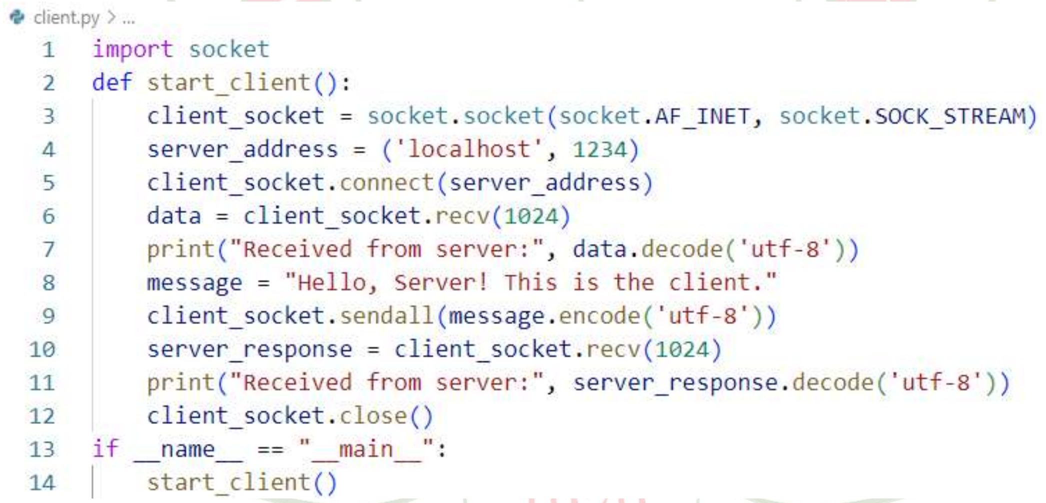
⇨ Program to demonstrate Socket Programming.

Server



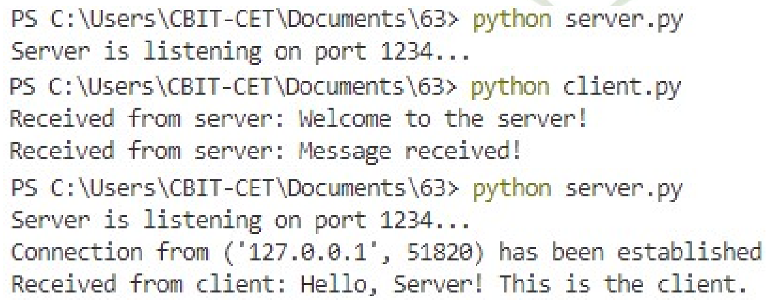
```python
import socket
def start_server():
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_address = ('localhost', 1234)
    server_socket.bind(server_address)
    server_socket.listen(1)
    print("Server is listening on port 1234...")
    while True:
        client_socket, client_address = server_socket.accept()
        print(f"Connection from {client_address} has been established.")
        welcome_message = "Welcome to the server!"
        client_socket.sendall(welcome_message.encode('utf-8'))
        client_message = client_socket.recv(1024)
        print("Received from client:", client_message.decode('utf-8'))
        response_message = "Message received!"
        client_socket.sendall(response_message.encode('utf-8'))
        client_socket.close()
if __name__ == "__main__":
    start_server()
```

Client



```python
import socket
def start_client():
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_address = ('localhost', 1234)
    client_socket.connect(server_address)
    data = client_socket.recv(1024)
    print("Received from server:", data.decode('utf-8'))
    message = "Hello, Server! This is the client."
    client_socket.sendall(message.encode('utf-8'))
    server_response = client_socket.recv(1024)
    print("Received from server:", server_response.decode('utf-8'))
    client_socket.close()
if __name__ == "__main__":
    start_client()
```

OUTPUT



```
PS C:\Users\CBIT-CET\Documents\63> python server.py
Server is listening on port 1234...
PS C:\Users\CBIT-CET\Documents\63> python client.py
Received from server: Welcome to the server!
Received from server: Message received!
PS C:\Users\CBIT-CET\Documents\63> python server.py
Server is listening on port 1234...
Connection from ('127.0.0.1', 51820) has been established
Received from client: Hello, Server! This is the client.
```