TERMWORK - 2 DATE :

Title of the Experiment:

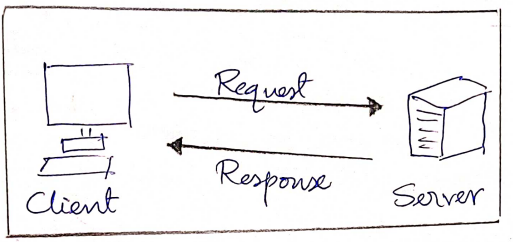
Implementing client server communication using socket programming that uses connection oriented protocol at transport layer.

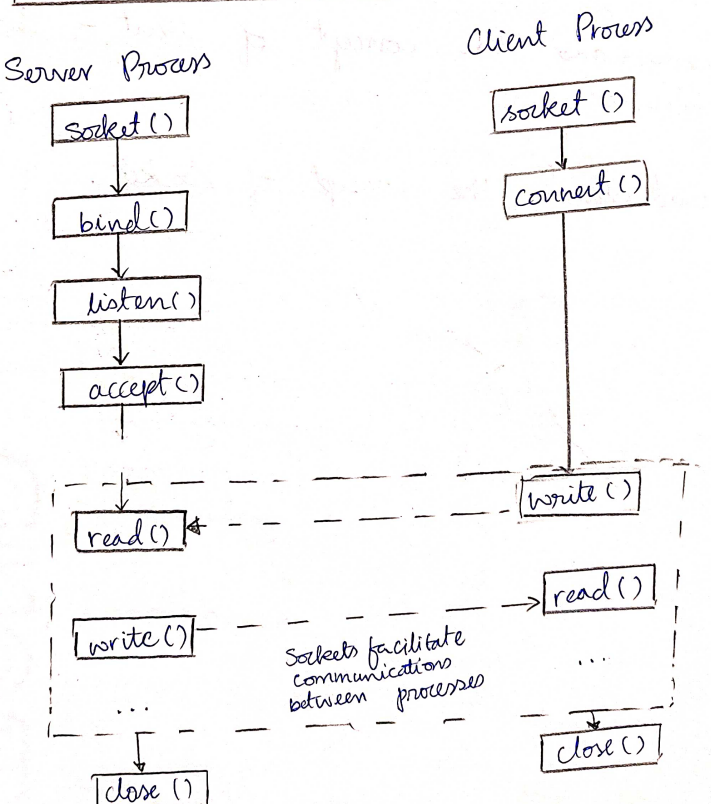
Objectives :

1) To understand the concept of Client - Server Communication.

2) To understand the concept of Socket.

Client Server Communication





State diagram for server and client model of socket.

Theory:

Client Server Communication:

- Clients and servers exchange messages in a request response messaging pattern.
- The client sends a request and the server returns a response.
- This exchange of message is an ~~exp~~ example of inter process communication.

- Examples: Mail servers, File servers, Web servers.
- Socket programming is a way of connecting two nodes on a network to communicate with each other.
- One socket (node) listens on a particular port at an IP, while other socket reaches out to the other to form a connection.
- Server forms the listener socket while client reaches out to the server.

Stages for server

1. Saket creation

int sockid = socket(domain, type, protocol)

sockid: saket descriptor, an integer (like a file handle)

domain: integer, specifies communication domain

Type: communication type

protocol: Protocol value for Internet Protocol (IP) which is 0.

- AF_LOCAL as defined in POSIX standard for communication between processes on the same host.
- AF_INET, AF_INET6 for processes on different hosts connected by IPv4

and for processes connected by IPV6.

2. Setsockopt: -Helps in manipulating options for the socket referred by the file descriptor sockfd.
This is completely optional but it helps in reuse of address and port. Prevents error such as: <address already in use>
int setsockopt(int sockfd, int level, int optname, const void *optv
Socklen_t optlen);

3. Bind:
int bind(int sockfd, const struct sockaddr *addr, socklen_t addrl
After creation of the socket, bind function binds the socket to t
address and port number specified

4. Listen:
int listen(int sockfd, int backlog);
It puts the server socket in a passive mode, where it waits for the client to approach the server to make a connection.

5. Accept:
int new_socket = accept(int sockfd, struct sockaddr *addr,
socklen_t *addrlen);
It extracts the first connection request on the queue of pending connections for the listening.

TCP:
- TCP is suited for applications that require high reliability and transmission time is relatively less critical.
- TCP rearranges data packets in the order specified
- TCP handles reliability and congestion control.
- TCP does flow control.
- Error checking and error recovery.

Algorithm: SERVER

1. Start
2. Create a socket
3. If socket creation failed
   Write Error message
   Goto 11
4. Assign IP and Port
5. Bind the created socket to IP
6. If socket bind fails
   Write Error message
   Goto 11
7. Accept the data packet from client
8. If accept fails
   Write Error
   Goto 11
9. While True:
   Read the message from client and copy it in buffer. Write buffer which contains the client contents. Copy server message in the buffer and send that buffer to client.
   If msg contains exit
   End the chat
   Break
10. Close the socket
11. Stop

Algorithm: CLIENT

1. Start
2. Create a socket.
3. If socket creation failed.
   Write Error message
   Goto 9
4. Assign IP and Port.
5. Connect the created client socket to server.
6. If connection fails
   Write Error message
   goto 9
7. While True:
   Read message
   Write message to socket
   Read message from server through socket. Write message to user if message is exit.
   Exit
   Break
8. Close the Socket
9. Stop

```
//TCPCLIENT.C
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define PORT 4444
void main(){

  int clientSocket;
  struct sockaddr_in serverAddr;
  char buffer[1024];

  clientSocket = socket(PF_INET, SOCK_STREAM, 0);
  printf("[+]Client Socket Created Sucessfully.\n");
  memset(&serverAddr, '\0', sizeof(serverAddr));
  serverAddr.sin_family = AF_INET;
  serverAddr.sin_port = htons(PORT);
  serverAddr.sin_addr.s_addr = inet_addr("127.0.0.1");

  connect(clientSocket, (struct sockaddr*)&serverAddr,
sizeof(serverAddr));
  printf("[+]Connected to Server.\n");
  recv(clientSocket, buffer, 1024, 0);
  printf("[+]Data Recv: %s\n", buffer);
  printf("[+]Closing the connection.\n");

}
```

```
//TCPSERVER.C

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

#define PORT 4444
void main(){

  int sockfd;
  struct sockaddr_in serverAddr;

  int newSocket;
  struct sockaddr_in newAddr
  socklen_t addr_size;
  char buffer[1024];

  sockfd = socket(AF_INET, SOCK_STREAM, 0);
  printf("[+]Server Socket Created Sucessfully.\n");
  memset(&serverAddr, '\0', sizeof(serverAddr));

  serverAddr.sin_family = AF_INET;
  serverAddr.sin_port = htons(PORT);
  serverAddr.sin_addr.s_addr = inet_addr("127.0.0.1");
  bind(sockfd, (struct sockaddr*)&serverAddr, sizeof(serverAddr));
  printf("[+]Bind to Port number %d.\n", 4455);

  listen(sockfd, 5);
  printf("[+]Listening...\n");

  newSocket = accept(sockfd, (struct sockaddr*)&newAddr, &addr_size);
  strcpy(buffer, "Hello");
  send(newSocket, buffer, strlen(buffer), 0);
  printf("[+]Closing the connection.\n");
}
```

Conclusion:

We successfully implemented client-server communication using socket programming that uses connection oriented protocol at transport layer.

Outcomes:

- We understood the concept of Client-Server Communication.
- We understood the concept of Sockets.

References:

W. Richard Stevens, Bill Fenner, Andrew M. Rodoff:
"UNIX Network Programming" Volume 1, Third edition, Pearson 2004.