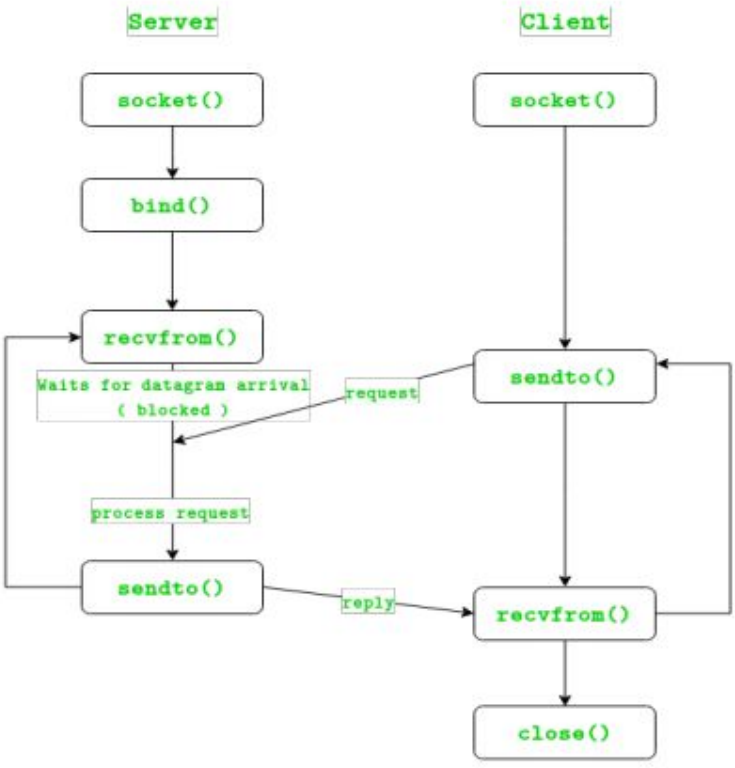
Experiment 1

AIM: Implement concurrent day-time client-server application

THEORY: UDP (User Datagram Protocol) is a communications protocol that is primarily used for establishing low-latency and loss-tolerating connections between applications on the internet. It speeds up transmissions by enabling the transfer of data before an agreement is provided by the receiving party. Unlike TCP, it is unreliable and connectionless protocol. So, there is no need to establish connection prior to data transfer.

Client–server model is a distributed application structure that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients. A server host runs one or more server programs, which share their resources with clients. A client does not share any of its resources, but it requests content or service from a server. The server component provides a function or service to one or many clients, which initiate requests for such services.



ALGORITHM

Server Socket

1. Create a socket - Get the file descriptor
2. Bind to an address
3. Listen on a port, and wait for a connection to be established.
4. Accept the connection from a client.
5. Send/ receive - the same way we read and write for a file.
6. Shutdown to end read/write.
7. Close to release data.

Client Socket

1. Create a socket.
2. Bind* -optional because you're the client, not the server.
3. Connect to a server.
4. Send/receive - repeat until we have or receive data
5. Shutdown to end read/write.
6. Close to release data.

SOURCE CODE:

Server

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#define PORT 8080
int main()
{
 int sockfd;
 char buffer[1024];
 struct sockaddr_in server_addr, client_addr;
 char time_str[1024];
 if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
 {
 perror("socket creation failed");
 exit(EXIT_FAILURE);
 }
```

```
 memset(&server_addr, 0, sizeof(server_addr));
 memset(&client_addr, 0, sizeof(client_addr));
 server_addr.sin_family = AF_INET;
 server_addr.sin_addr.s_addr = INADDR_ANY;
 server_addr.sin_port = htons(PORT);
 if (bind(sockfd, (const struct sockaddr *)&server_addr,
 sizeof(server_addr)) < 0)
 {
 perror("bind failed");
 exit(EXIT_FAILURE);
 }
 else
 {
 printf("\nSocket binded.\n");
 }
while(1){
time_t t = time(NULL);
struct tm tm = *localtime(&t);
snprintf(time_str, sizeof(time_str), "Now: %d-%02d-%02d %02d:%02d:%02d\n",
tm.tm_year + 1900, tm.tm_mon + 1, tm.tm_mday, tm.tm_hour, tm.tm_min, tm.tm_sec);
 int n, len, max_len = 1024;
 len = sizeof(client_addr);
 n = recvfrom(sockfd, (char *)buffer, max_len,
 MSG_WAITALL, (struct sockaddr *)&client_addr,&len);
 buffer[n] = '\0';
 printf("Client : %s\n", buffer);
 sendto(sockfd, (const char *)time_str, strlen(time_str),
 MSG_CONFIRM, (const struct sockaddr *)&client_addr, len);
 printf("Message sent.\n");
 }
 return 0;
}
```

Client

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
```

```
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#define PORT 8080
int main()
{
 int sockfd;
 char buffer[1024];
 char const *request = "Request from client";
 struct sockaddr_in server_addr;
 if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
{
 perror("socket creation failed");
 exit(EXIT_FAILURE);
 }
 memset(&server_addr, 0, sizeof(server_addr));
 server_addr.sin_family = AF_INET;
 server_addr.sin_port = htons(PORT);
 server_addr.sin_addr.s_addr = INADDR_ANY;
 int n, len, ch = 1, max_len = 1024;
 while (ch == 1)
 {
 sendto(sockfd, (const char *)request, strlen(request),
 MSG_CONFIRM, (const struct sockaddr *)&server_addr, sizeof(server_addr));
 printf("Request sent.\n");
 n = recvfrom(sockfd, (char *)buffer, max_len,
 MSG_WAITALL, (struct sockaddr *)&server_addr,&len);
 buffer[n] = '\0';
 printf("Server : %s\n", buffer);
 printf("Want to send another request? (Yes(1)/No(0))");
 scanf("%d", &ch);
 }
 close(sockfd);
 printf("\nClient socket closed.\n");
 return 0;
}
```

OUTPUT:

Server

```
kunal@DESKTOP-AITAEP7:/mnt/c/Users/Admin/Desktop/college/7th Semester/Distributed System/LAb
$ gcc daytimeserver.c -o server
kunal@DESKTOP-AITAEP7:/mnt/c/Users/Admin/Desktop/college/7th Semester/Distributed System/LAb
$ ./server

Socket binded.
Client : Request from client
Message sent.
Client : Request from client
Message sent.
Client : Request from client
Message sent.
```

Client

```
kunal@DESKTOP-AITAEP7:/mnt/c/Users/Admin/Desktop/college/7th Semester/Distributed System/LAb
$ gcc daytimeclient.c -o client
kunal@DESKTOP-AITAEP7:/mnt/c/Users/Admin/Desktop/college/7th Semester/Distributed System/LAb
$ ./client
Request sent.
Server : Now: 2020-09-14 00:32:22

Want to send another request? (Yes(1)/No(0))1
Request sent.
Server : Now: 2020-09-14 00:34:00

Want to send another request? (Yes(1)/No(0))1
Request sent.
Server : Now: 2020-09-14 00:34:04

Want to send another request? (Yes(1)/No(0))0

Client socket closed.
```

LEARNING OUTCOMES:

We learnt how to implement a Client-Server Communication using UDP protocol and also about socket programming. We also learned how it is inefficient as there is no reliability of connection.