Exp No. :3.B

Exp Name: TCP socket programming (Multiclient using threading)

Problem Statement: Create a multi-client TCP server. The client will take a dataword and divisor from the user and send them to the server. The server will find out the codeword using CRC and send it to the client.

Theory:

Client-server networks are computer networks that employ a dedicated computer to store data, manage/provide resources, and control user access (server). The server connects all of the other computers in the network by acting as a hub. A machine that connects to the server is known as a client.

A multiple client server is a type of software architecture for computer networks where clients, which can be basic workstations or fully functional personal computers, request information from a server computer.

The simple way to handle multiple clients would be to spawn new thread for every new client connected to the server.

A thread is a sequential flow of tasks within a process. Each thread has its own set of registers and stack space. There can be multiple threads in a single process having the same or different functionality. Threads are a relatively lightweight way to implement multiple paths of execution inside of an application. At the system level, programs run side by side, with the system doling out execution time to each program based on its needs and the needs of other programs.

On a multiprocessor system, multiple threads can concurrently run on multiple CPUs. Therefore, multithreaded programs can run much faster than on a uniprocessor system. They can also be faster than a program using multiple processes, because **threads require fewer resources and generate less overhead**.

A thread has the following three components:

- Program Counter
- Register Set
- Stack space

Threads in the operating system provide multiple benefits and improve the overall performance of the system. Some of the reasons threads are needed in the operating system are:

- Since threads use the same data and code, the operational cost between threads is low.
- Creating and terminating a thread is faster compared to creating or terminating a process.
- Context switching is faster in threads compared to processes.

In Multithreading, the idea is to divide a single process into multiple threads instead of creating a whole new process. Multithreading is done to achieve parallelism and to improve the performance

of the applications as it is faster in many ways which were discussed above. The other advantages of multithreading are mentioned below.

- **Resource Sharing:** Threads of a single process share the same resources such as code, data/file.

- **Responsiveness:** Program responsiveness enables a program to run even if part of the program is blocked or executing a lengthy operation. Thus, increasing the responsiveness to the user.
- **Economy:** It is more economical to use threads as they share the resources of a single process. On the other hand, creating processes is expensive.

Thread functions in C/C++ In a Unix/Linux operating system, the C/C++ languages provide the POSIX thread(pthread) standard API(Application program Interface) for all thread related functions. It allows us to create multiple threads for concurrent process flow.

CODE:

Server:

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

#define MAXCLIENT 3
int server_sockfd;

void *serve(void *arg)
{
      char ch;
      int client_sockfd,client_len;
      struct sockaddr_in client_address;
      printf("server waiting\n");
      client_len = sizeof(client_address);
      client_sockfd = accept(server_sockfd,(struct
sockaddr*)&client_address, &client_len);
      read(client_sockfd, &ch, 1);
      ch++;
      write(client_sockfd, &ch, 1);
      return NULL;
}

int main(){
      int client_sockfd, server_sockfd;
      int server_len, client_len,temp;
      struct sockaddr_in server_address;
      pthread_t th[MAXCLIENT];
      server_sockfd = socket(AF_INET, SOCK_STREAM, 0);
      server_address.sin_family = AF_INET;
      server_address.sin_addr.s_addr = inet_addr("127.0.0.1");
      server_address.sin_port = 9734;
      server_len = sizeof(server_address);
      bind(server_sockfd, (struct sockaddr*)&server_address, server_len);
      listen(server_sockfd, MAXCLIENT);
      printf("Server started.\n");
      temp=MAXCLIENT;
      while(1)
            pthread_create(&th[1], NULL, serve, NULL);
      temp=MAXCLIENT;
      while(1)
            pthread_join(th[1],NULL);
```

```
      return 0;
}
```

CLIENT:

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

int main()
{
      int sockfd;
      int len;
      struct sockaddr_in address;
      int result;
      char ch = 'A';

      sockfd = socket(AF_INET, SOCK_STREAM, 0);

      address.sin_family = AF_INET;
      address.sin_addr.s_addr = inet_addr("127.0.0.1");
      address.sin_port = 9734;
      len = sizeof(address);

      result = connect(sockfd, (struct sockaddr *)&address, len);


      if(result == -1){
            perror("oops: client1");
            exit(1);
      }

      write(sockfd, &ch, 1);
      read(sockfd, &ch, 1);

      printf("Char from Server:  %c\n", ch);
      close(sockfd);

      return 0;
}
```

OUTPUT:

CONCLUSION:

Difference between Process and thread
Advantages and Disadvantages of thread
other ways to implement multi-client server