Distributed Systems (CS-304)

Assignment-4

NAME: SOURABH PATEL

ADMISSION NO: U19CS082

Implement given extensions to the Client Server Programming.

1. Extend your echo Client Server message passing application to chat application.

● Client and Server are able to send the message to each other until one of them quits or terminates.

SERVER:

```
/*
SERVER
*/
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h>
#include <unistd.h>
#define MAXLINE 4096 /*max text line length*/
#define SERV_PORT 3000 /*port*/
#define LISTENQ 8 /*maximum number of client connections */
int main (int argc, char **argv)
{
    int listenfd, connfd, n;
    socklen_t clilen;
    char buf[MAXLINE];
    struct sockaddr_in cliaddr, servaddr;

    //creation of the socket
    listenfd = socket (AF_INET, SOCK_STREAM, 0);
    //preparation of the socket address
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servaddr.sin_port = htons(SERV_PORT);
    bind (listenfd, (struct sockaddr *) &servaddr, sizeof(servaddr));
    listen (listenfd, LISTENQ);
    printf("%s\n","Server running...waiting for connections.");
```

```
    for ( ; ; ) {
        clilen = sizeof(cliaddr);
        connfd = accept (listenfd, (struct sockaddr *) &cliaddr, &clilen);
        printf("%s\n","Received request...");
        while ( (n = recv(connfd, buf, MAXLINE,0)) > 0) {
            printf("%s","String received from and resent to the client:");
            puts(buf);
            send(connfd, buf, n, 0);
        }
        if (n < 0) {
            perror("Read error");
            exit(1);
        }
        close(connfd);
    }
    //close listening socket
    close (listenfd);
}
```

```
~$ gcc server.c -o server
~$ ./server
Server running...waiting for connections.
Received request...
String received from and resent to the client:hi

String received from and resent to the client:i am

String received from and resent to the client:ok by
```

CLIENT:

```
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h>
#include <arpa/inet.h>
#define MAXLINE 4096 /*max text line length*/
#define SERV_PORT 3000 /*port*/
int main(int argc, char **argv)
{
    int sockfd;
    struct sockaddr_in servaddr;
    char sendline[MAXLINE], recvline[MAXLINE];

    //Create a socket for the client
```

```
    //If sockfd<0 there was an error in the creation of the socket
    if ((sockfd = socket (AF_INET, SOCK_STREAM, 0)) <0) {
        perror("Problem in creating the socket");
        exit(2);
    }

    //Creation of the socket
    memset(&servaddr, 0, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr= inet_addr("127.0.0.1");
    servaddr.sin_port = htons(SERV_PORT); //convert to big-endian order

    //Connection of the client to the socket
    if (connect(sockfd, (struct sockaddr *) &servaddr, sizeof(servaddr))<0) {
        perror("Problem in connecting to the server");
        exit(3);
    }
    while (fgets(sendline, MAXLINE, stdin) != NULL) {
        send(sockfd, sendline, strlen(sendline), 0);
        if (recv(sockfd, recvline, MAXLINE,0) == 0){
            //error: server terminated prematurely
            perror("The server terminated prematurely");
            exit(4);
        }
        printf("%s", "String received from the server: ");
        fputs(recvline, stdout);
    }
    exit(0);
}
```

```
~$ gcc client.c -o client
~$ ./client
hi
String received from the server: hi
String received from the server: i am
ok bye!
String received from the server: ok by
```

2. Using the Client-Server communication mechanism get the load status of other nodes in your network (identify the states of other nodes in the system – Overload, Moderate, Lightly). ● Implement the Client-Server model. Run the client and server instance on same machine and pass the message from client to server or server to client ● Get the CPU load of the client or server and state that either it is under loaded or overloaded.

SERVER:

```
#include<stdio.h>
#include<netinet/in.h>
#include<netdb.h>
#include<sys/types.h>
#include<sys/stat.h>
#include <unistd.h>
#define SERV_TCP_PORT 5035
int main(int argc,char**argv)
{
    int sockfd, newsockfd, clength;
    struct sockaddr_in serv_addr, cli_addr;
    char buffer[4096];

    //Create socket
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    puts("Socket created");

    //Prepare the sockaddr_in structure
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = INADDR_ANY;
    serv_addr.sin_port = htons(SERV_TCP_PORT);

    //Bind
    printf("\nStart");
    bind(sockfd,(struct sockaddr*)&serv_addr,sizeof(serv_addr));
    printf("\nListening...");
    printf("\n");
    listen(sockfd, 5);

    //accept connection from an incoming client
    clength = sizeof(cli_addr);
    newsockfd = accept(sockfd, (struct sockaddr*)&cli_addr, &clength);
    printf("\nAccepted");
```

```
    printf("\n");

    //print message from client
    read(newsockfd, buffer, 4096);
    printf("\nClient message:%s", buffer);
    write(newsockfd,buffer, 4096);
    printf("\n");

    //close the socket
    close(sockfd);
    return 0;
}
```

```
~$ gcc proServ.c -o pro
~$ ./pro
Socket created

Start
Listening...

Accepted

Client message:Lightly-loaded
```

CLIENT:

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include <arpa/inet.h>
#include<netinet/in.h>
#include<netdb.h>
#include <unistd.h>
#define SERV_TCP_PORT 5035
int main(int argc,char*argv[])
{
    int sockfd;
    struct sockaddr_in serv_addr;
    struct hostent *server;
    char buffer[4096];

    //Create socket
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
    serv_addr.sin_port = htons(SERV_TCP_PORT);
```

```c
    //Connect to remote server
    printf("\nReady for sending...");
    connect(sockfd,(struct sockaddr*)&serv_addr, sizeof(serv_addr));
    printf("\nClient Information: ");
    //fgets(buffer, 4096, stdin);

    FILE *pp;
    pp = popen("./findUsage.sh", "r");
    if (pp != NULL) {
        while (1) {
            char *line;
            line = fgets(buffer, sizeof buffer, pp);
            if (line == NULL) break;
            printf("%s", line); /* line includes '\n' */
        }
        pclose(pp);
    }
    //send to server
    write(sockfd,buffer,4096);
    printf("Serverecho:%s",buffer);
    printf("\n");
    close(sockfd);
    return 0;
}
```

```
~$ gcc proClient.c -o proc
~$ ./proc

Ready for sending...
Client Information: Total CPU Usage : 23.1%
Lightly-loaded
Serverecho:Lightly-loaded
```