Thangal Kunju Musaliar College of Engineering

Kollam, Kerala

NETWORKING LAB RECORD

Department of Computer Science and Engineering

MAY 2024

Thangal Kunju Musaliar College of Engineering

Kollam, Kerala

NETWORKING LAB RECORD

CERTIFICATE

Certified that this is a bonafide record of the work done by … ..... of …….. Semester, Class Roll No .....in the…… Laboratory during the year ……

Reg.no : ……..

Name of the Examination: **……..**

Staff member in charge *External Examiner*

VISION

To be a centre of excellence imparting quality education in Computer Science and Engineering and transforming students to critical thinkers and lifelong learners capable of developing environment friendly and economically feasible solutions to real world problems.

MISSION

- To provide strong foundation in Computer Science and Engineering, prepare students for professional career and higher education, and inculcate research interest.
- To be abreast of the technological advances in a rapidly changing world.
- To impart skills to come up with socially acceptable solutions to real world problems, upholding ethical values.

PROGRAMME EDUCATIONAL OBJECTIVES(PEOS)

PEO 1: Excel in professional career by acquiring knowledge in mathematics, science, and engineering and applying the knowledge in the design of hardware and software Solutions for challenging problems of the society.

PEO 2: Pursue higher studies and research thereby engages in lifelong learning by adapting to the current trends in the area of Computer Science & Engineering.

PEO 3: Ability to provide socially acceptable and economically feasible computer oriented solutions to real world problems with teamwork, while maintaining environmental balance, quality and cognizance of the underlying principles of ethics.

PROGRAM SPECIFIC OUTCOMES (PSOs)

PSO1: Apply mathematical and algorithmic principles, data structure concepts, software, and hardware and techniques in designing and developing optimized and secure computer based solutions.

PSO2: Design and develop system software and provide exposure to various tools and programming languages to facilitate efficient computing environment which adds to the ease of human life.

PSO3: Use the knowledge of various data processing, communication and intelligent systems to provide solutions to new ideas and innovations.

PROGRAMME OUTCOMES

1. Engineering knowledge : Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

2. Problem analysis : Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first Principles of mathematics, natural sciences, and engineering sciences.

3. Design/development of solutions : Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

4. Conduct investigations of complex problems : Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5. Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

6. The engineer and society : Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

7. Environment and sustainability : Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

8. Ethics : Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. Individual and team work : Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. Communication : Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. Project management and finance : Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

12. Life-long learning : Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

COURSE OUTCOMES

CO1	Use network related commands and configuration files in Linux Operating System. **(Cognitive Knowledge Level: Understand).**
CO2	Develop network application programs and protocols. **(Cognitive Knowledge Level: Apply)**
CO3	Analyze network traffic using network monitoring tools. **(Cognitive Knowledge Level: Apply)**
CO4	Design and setup a network and configure different network protocols. **(Cognitive Knowledge Level: Apply)**
CO5	Develop simulation of fundamental network concepts using a network simulator. **(Cognitive Knowledge Level: Apply)**

INDEX

SI NO.	EXPERIMENT NAME	PAGE NO	DATE
	CYCLE 1		
1	FAMILIARISATION OF NETWORKING COMMANDS IN LINUX		
2	FAMILIARISATION OF NETWORKING COMMANDS IN WINDOWS		
3	SYSTEM CALLS IN SOCKET PROGRAMMING		
4	FINDING MINIMUM,MAXIMUM AND AVERAGE OF INTEGER ARRAY USING SOCKET PROGRAMMING		
5	TWO CLIENT ONE SERVER SYSTEMS		
6	MULTI USER CHAT SERVER USING TCP		
7	CONCURRENT TIME SERVER APPLICATION USING UDP		
	CYCLE 2		
1	FLOW CONTROL PROTOCOLS		
a	STOP-AND-WAIT ARQ		
b	GO-BACK N ARQ		
c	SELECTIVE REPEAT ARQ		
2	DISTANCE VECTOR ROUTING ALGORITHM		
3	LINK STATE ROUTING ALGORITHM		
4	FILE TRANSFER PROTOCOL		
5	CONCURRENT FILE SERVER		
6	LEAKY BUCKET ALGORITHM		

IMPLEMENTATION

```
/* NAME : BHAGYA A JAI
   ROLL NO : B21CSB18
   EXPERIMENT : SOCKET PROGRAMMING FOR DATA TRANSFER*/
```

TCP SERVER

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<unistd.h>
#include<arpa/inet.h>
int main()
{
        char *ip="127.0.0.1";
        int port=5566;
        int server_sock,client_sock;
        struct sockaddr_in server_addr,client_addr;
        socklen_t addr_size;
        char buffer[1024];
        int n;
        server_sock=socket(AF_INET,SOCK_STREAM,0);
        if (server_sock<0)
        {
                perror("[-]Socket error");
                exit(1);
        }
        printf("[+]TCP server socket created.\n");
        memset(&server_addr,'\0',sizeof(server_addr));
        server_addr.sin_family=AF_INET;
        server_addr.sin_port=port;
        server_addr.sin_addr.s_addr=inet_addr(ip);
        n=bind(server_sock,(struct sockaddr*)&server_addr,sizeof(server_addr));
        if(n<0)
        {
                perror("[-]Bind error");
                exit(1);
        }
        printf("[+]Bind to the port number: %d\n",port);
        listen(server_sock,5);
        printf("Listening....\n");
        while(1)
        {
                addr_size=sizeof(client_addr);
                client_sock=accept(server_sock,(struct sockaddr*)&client_addr,&addr_size);
                printf("[+]Client Connected.\n");
                bzero(buffer,1024);
                recv(client_sock,buffer,sizeof(buffer),0);
                printf("Client:%s\n",buffer);
                bzero(buffer,1024);
                strcpy(buffer,"HI THIS IS THE SERVER!!");
                printf("Server:%s\n",buffer);
                send(client_sock,buffer,strlen(buffer),0);
```

```
                close(client_sock);
                printf("[+]Client disconnected.\n\n");
        }
        close(server_sock);
        return 0;
}
```

TCP CLIENT

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<unistd.h>
#include<arpa/inet.h>
int main()
{
        char *ip="127.0.0.1";
        int port=5566;
        int sock;
        struct sockaddr_in addr;
        socklen_t addr_size;
        char buffer[1024];
        int n;
        sock=socket(AF_INET,SOCK_STREAM,0);
        if (sock<0)
        {
                perror("[-]Socket error");
                exit(1);
        }
        printf("[+]TCP client socket created.\n");
        memset(&addr,'\0',sizeof(addr));
        addr.sin_family=AF_INET;
        addr.sin_port=port;
        addr.sin_addr.s_addr=inet_addr(ip);
        connect(sock,(struct sockaddr*)&addr,sizeof(addr));
        printf("Connected to the Server.\n");
        bzero(buffer,1024);
        strcpy(buffer,"HELLO,THIS IS THE CLIENT");
        printf("Client:%s\n",buffer);
        send(sock,buffer,strlen(buffer),0);
        bzero(buffer,1024);
        recv(sock,buffer,sizeof(buffer),0);
        printf("Server:%s\n",buffer);
        close(sock);
        printf("Disconnected from the Server.\n");
        return 0;
}
```

UDP SERVER

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/socket.h>
#include <arpa/inet.h>
```

```
#define PORT 8000
void main() {
   int server_fd;
   struct sockaddr_in serv_addr, cli_addr;
   char buff[6];
   printf("UDP Server\n");
   if ((server_fd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
      printf("Socket creation failed!\n");
      exit(1);
   } else {
      printf("Server socket created.\n");
   }
   serv_addr.sin_family = AF_INET;
   serv_addr.sin_addr.s_addr = INADDR_ANY; // OR inet_addr("172.20.34.12")
   serv_addr.sin_port = htons(PORT);
   if (bind(server_fd, (struct sockaddr*) &serv_addr, sizeof(serv_addr)) < 0) {
      printf("Binding failed!\n");
      exit(1);
   } else {
      printf("Socket binded.\n");
   }
   int len = sizeof(cli_addr);
   if (recvfrom(server_fd, (char*) buff, 6 * sizeof(char), 0, (struct sockaddr*) &cli_addr, &len) < 0)
{
      printf("Receive failed!\n");
      exit(1);
   } else {
      printf("Received message: %s\n", buff);
   }
   close(server_fd);
}
```

UDP CLIENT

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#define PORT 8000
void main() {
   int client_fd;
   struct sockaddr_in serv_addr;
   char buff[] = "HELLO";
   printf("UDP Client\n");
   if ((client_fd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
      printf("Socket creation failed!\n");
      exit(1);
   } else {
      printf("Client socket created.\n");
   }
   serv_addr.sin_family = AF_INET;
   serv_addr.sin_addr.s_addr = INADDR_ANY; // OR inet_addr("172.20.34.12")
   serv_addr.sin_port = htons(PORT);
```

```
    if (sendto(client_fd, buff, 6 * sizeof(char), 0, (struct sockaddr*) &serv_addr, sizeof(serv_addr)) <
0) {
        printf("Sending failed!\n");
        exit(1);
    } else {
        printf("Message sent: %s\n", buff);
    }
    close(client_fd);
}
```

OUTPUT

TCP

```
root@DESKTOP-70V9DEP:~# gcc server.c
root@DESKTOP-70V9DEP:~# ./a.out
[+]TCP server socket created.
[+]Bind to the port number: 5566
Listening....
[+]Client Connected.
Client:HELLO,THIS IS THE CLIENT
Server:HI THIS IS THE SERVER!!
[+]Client disconnected.

root@DESKTOP-70V9DEP:~# gcc client.c
root@DESKTOP-70V9DEP:~# ./a.out
[+]TCP client socket created.
Connected to the Server.
Client:HELLO,THIS IS THE CLIENT
Server:HI THIS IS THE SERVER!!
Disconnected from the Server.
```

UDP

```
root@DESKTOP-70V9DEP:~# gcc udps1.c
root@DESKTOP-70V9DEP:~# ./a.out
UDP Server
Server socket created.
Socket binded.
Received message: HELLO

root@DESKTOP-70V9DEP:~# gcc udpc1.c
root@DESKTOP-70V9DEP:~# ./a.out
UDP Client
Client socket created.
Message sent: HELLO
```

IMPLEMENTATION

```
/* NAME : BHAGYA A JAI
   ROLL NO : B21CSB18
   EXPERIMENT : ARRAY OPERATIONS USING SOCKET PROGRAMMING*/
```

TCP SERVER

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<unistd.h>
#include<arpa/inet.h>
#define PORT 5568
#define MAX_SIZE 100
void calculate_statistics(int arr[], int size, int *min, int *max, float *avg) {
   *min = arr[0];
   *max = arr[0];
   *avg = 0;
   for (int i = 0; i < size; i++) {
      if (arr[i] < *min)
         *min = arr[i];
      if (arr[i] > *max)
         *max = arr[i];
      *avg += arr[i];
   }
   *avg /= size;
}
int main() {
   char *ip = "127.0.0.1";
   int server_sock, client_sock;
   struct sockaddr_in server_addr, client_addr;
   socklen_t addr_size;
   int n;
   int received_array[MAX_SIZE];
   int array_size;
   server_sock = socket(AF_INET, SOCK_STREAM, 0);
   if (server_sock < 0) {
      perror("[-]Socket error");
      exit(1);
   }
   printf("[+]TCP server socket created.\n");
   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 = inet_addr(ip);
   n = bind(server_sock, (struct sockaddr*)&server_addr, sizeof(server_addr));
   if (n < 0) {
      perror("[-]Bind error");
      exit(1);
   }
   printf("[+]Bind to the port number: %d\n", PORT);
   listen(server_sock, 5);
   printf("Listening....\n");
```

```
    while (1) {
        addr_size = sizeof(client_addr);
        client_sock = accept(server_sock, (struct sockaddr*)&client_addr, &addr_size);
        printf("[+]Client Connected.\n");
        recv(client_sock, &array_size, sizeof(int), 0);
        recv(client_sock, received_array, array_size * sizeof(int), 0);
        int min, max;
        float avg;
        calculate_statistics(received_array, array_size, &min, &max, &avg);
        send(client_sock, &min, sizeof(int), 0);
        send(client_sock, &max, sizeof(int), 0);
        send(client_sock, &avg, sizeof(float), 0);
        close(client_sock);
        printf("[+]Client disconnected.\n\n");

        break;
    }
    close(server_sock);
    return 0;
}
```

TCP CLIENT

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<unistd.h>
#include<arpa/inet.h>
#define PORT 5568
#define MAX_SIZE 100
int main() {
    char *ip = "127.0.0.1";
    int client_sock;
    struct sockaddr_in server_addr;
    int n;
    int array[MAX_SIZE];
    int array_size;
    int min, max;
    float avg;
    client_sock = socket(AF_INET, SOCK_STREAM, 0);
    if (client_sock < 0) {
        perror("[-]Socket error");
        exit(1);
    }
    printf("[+]TCP client socket created.\n");
    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 = inet_addr(ip);
    connect(client_sock, (struct sockaddr*)&server_addr, sizeof(server_addr));
    printf("Connected to the Server.\n");

    printf("Enter the size of the array: ");
    scanf("%d", &array_size);
```

```
    printf("Enter the elements of the array: ");
    for (int i = 0; i < array_size; i++) {
        scanf("%d", &array[i]);
    }
    send(client_sock, &array_size, sizeof(int), 0);
    send(client_sock, array, array_size * sizeof(int), 0);
    recv(client_sock, &min, sizeof(int), 0);
    recv(client_sock, &max, sizeof(int), 0);
    recv(client_sock, &avg, sizeof(float), 0);
    printf("Server: Minimum = %d, Maximum = %d, Average = %.2f\n", min, max, avg);
    close(client_sock);
    printf("Disconnected from the Server.\n");
    return 0;
}
```

UDP SERVER

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#define PORT 8000
float findMin(float *buff, int n) {
    int min = 0;
    for (int i = 0; i < n; i++) {
        if (buff[min] > buff[i])
            min = i;
    }
    return buff[min];
}
float findMax(float *buff, int n) {
    int max = 0;
    for (int i = 0; i < n; i++) {
        if (buff[max] < buff[i])
            max = i;
    }
    return buff[max];
}
float findAvg(float *buff, int n) {
    float avg = 0;
    for (int i = 0; i < n; i++) {
        avg += buff[i];
    }
    avg /= n;
    return avg;
}
void main() {
    int server_fd, client_fd;
    struct sockaddr_in address, cli_addr;
    int n, addrlen = sizeof(address);
    float *buff, output[3];
    printf("TCP Server\n");
```

```
    if ((server_fd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
        printf("Socket creation failed!\n");
        exit(1);
    } else {
        printf("Server socket created.\n");
    }
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(PORT);
    if (bind(server_fd, (struct sockaddr*) &address, addrlen) < 0) {
        printf("Socket binding failed!\n");
        exit(1);
    } else {
        printf("Socket binded.\n");
    }
    int len = sizeof(cli_addr);
    if (recvfrom(server_fd, &n, sizeof(int), 0, (struct sockaddr*) &cli_addr, &len) < 0) {
        printf("Receive failed!\n");
        exit(1);
    } else {
        buff = (float*) malloc(n * sizeof(float));
        if (recvfrom(server_fd, buff, n * sizeof(float), 0, (struct sockaddr*) &cli_addr, &len) < 0) {
            printf("Receive failed!\n");
            exit(1);
        } else {
            printf("Received data:");
            for (int i = 0; i < n; i++) {
                printf(" %.1f", buff[i]);
            }
            printf("\n");
        }
    }
    printf("Calculating min, max and avg.\n");
    output[0] = findMin(buff, n);
    output[1] = findMax(buff, n);
    output[2] = findAvg(buff, n);
    if (sendto(server_fd, output, 3 * sizeof(float), 0, (struct sockaddr*) &cli_addr, len) < 0) {
        printf("Send failed!\n");
        exit(1);
    } else {
        printf("Result sent to client.\n");
    }
    close(server_fd);
    close(client_fd);
}
```

UDP CLIENT

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
```

```
#define PORT 8000
void main() {
   int client_fd;
   struct sockaddr_in serv_addr;
   float *buff, reply[3];
   int n;
   printf("TCP Client\n");
   client_fd = socket(AF_INET, SOCK_DGRAM, 0);
   if (client_fd < 0) {
      printf("Socket creation failed!\n");
      exit(1);
   } else {
      printf("Client socket created.\n");
   }
   serv_addr.sin_family = AF_INET;
   serv_addr.sin_addr.s_addr = INADDR_ANY;
   serv_addr.sin_port = htons(PORT);
   printf("Enter size: ");
   scanf("%d", &n);
   buff = (float*) malloc(n * sizeof(float));
   printf("Enter numbers: ");
   for (int i = 0; i < n; i++)
      scanf("%f", &buff[i]);
   int len = sizeof(serv_addr);
   if (sendto(client_fd, &n, sizeof(int), 0, (struct sockaddr*) &serv_addr, len) < 0) {
      printf("Send failed!\n");
      exit(1);
   } else {
      if (sendto(client_fd, buff, n * sizeof(float), 0, (struct sockaddr*) &serv_addr, len) < 0) {
         printf("Send failed!\n");
         exit(1);
      } else {
         printf("Data sent:");
         for (int i = 0; i < n; i++) {
            printf(" %.1f", buff[i]);
         }
         printf("\nWaiting for reply.\n");
      }
   }
   if (recvfrom(client_fd, reply, 3 * sizeof(float), 0, (struct sockaddr*) &serv_addr, &len) < 0) {
      printf("Receive failed!\n");
      exit(1);
   } else {
      printf("Result received:\n");
      printf("Min: %.1f\n", reply[0]);
      printf("Max: %.1f\n", reply[1]);
      printf("Avg: %.1f\n", reply[2]);
   }
   close(client_fd);
}
```

OUTPUT

TCP

```
root@DESKTOP-70V9DEP:~# gcc serverminme.c
root@DESKTOP-70V9DEP:~# ./a.out
[+]TCP server socket created.
[+]Bind to the port number: 5568
Listening....
[+]Client Connected.
[+]Client disconnected.

root@DESKTOP-70V9DEP:~# gcc clientminme.c
root@DESKTOP-70V9DEP:~# ./a.out
[+]TCP client socket created.
Connected to the Server.
Enter the size of the array: 6
Enter the elements of the array: 5 2 8 4 9 1
Server: Minimum = 1, Maximum = 9, Average = 4.83
Disconnected from the Server.
```

UDP

```
root@DESKTOP-70V9DEP:~# gcc udps2.c
root@DESKTOP-70V9DEP:~# ./a.out
TCP Server
Server socket created.
Socket binded.
Received data: 5.5 3.4 4.2 2.0 1.1
Calculating min, max and avg.
Result sent to client.

root@DESKTOP-70V9DEP:~# gcc udpc2.c
root@DESKTOP-70V9DEP:~# ./a.out
TCP Client
Client socket created.
Enter size: 5
Enter numbers: 5.5 3.4 4.2 2.0 1.1
Data sent: 5.5 3.4 4.2 2.0 1.1
Waiting for reply.
Result received:
Min: 1.1
Max: 5.5
Avg: 3.2
```

IMPLEMENTATION

A)

```
/* NAME : BHAGYA A JAI
   ROLL NO : B21CSB18
   EXPERIMENT : REVERSE A STRING*/
```

SERVER

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<string.h>
#include<arpa/inet.h>
int main()
{
        int sesoc;
        struct sockaddr_in serveraddr;
        int newsoc;
        struct sockaddr_in newaddr; socklen_t addr_size;
        char buf[50],t;
        sesoc=socket(AF_INET,SOCK_STREAM,0);
        memset(&serveraddr,'\0',sizeof(serveraddr));
        if(sesoc==-1)
        {
                printf("socket creation failed");
        }
        else
        {
                serveraddr.sin_family=AF_INET;
                serveraddr.sin_port=htons(4955);
                serveraddr.sin_addr.s_addr=inet_addr("127.0.0.1");
                bind(sesoc,(struct sockaddr *)&serveraddr,
                sizeof(serveraddr));
                listen(sesoc,5); addr_size=sizeof(newsoc);
                newsoc=accept(sesoc,(struct sockaddr *)&newaddr,&addr_size);
                recv(newsoc,&buf,sizeof(buf),0); printf("SUCCESSFULLY RECEIVED \n");
                int n=strlen(buf);
                for(int i=0;i<n/2;i++)
                {
                        t=buf[i]; buf[i]=buf[n-i-1];
                        buf[n-i-1]=t;
                }
                int newsoc1 = accept(sesoc,(struct sockaddr*)&newaddr,&addr_size);
                int sid=send(newsoc1,buf,sizeof(buf),0);
                if(sid<0)
                {
                        printf("Reverse Unsuccessful\n");
                        exit(0);
                }
                printf("SENDING SUCCESSFUL\n");
        }
        return 0;
```

```
}
```
CLIENT1
```
#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>
int main(int argc,char *argv[])
{
	int clsoc; float avg;
	struct sockaddr_in claddr; char buf[1024];
	clsoc=socket(AF_INET,SOCK_STREAM,0);
	memset(&claddr,'\0',sizeof(claddr));
	if(clsoc==-1)
	{
		printf("error in creation\n");
	}
	else
	{
		claddr.sin_family=AF_INET;
		claddr.sin_port=htons(4955);
		claddr.sin_addr.s_addr=inet_addr("127.0.0.1");
		int conn_stat=connect(clsoc,(struct sockaddr*)&claddr,sizeof(claddr));
		if(conn_stat==-1)
		{
			printf("Connection error\n");
		}
		else
		{
			printf("Enter String: ");
			scanf("%[^\n]",buf);
			//printf("%s",buf);
			send(clsoc,buf,sizeof(buf),0);
		}
		return 0;
	}
}
```
CLIENT2
```
#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>
int main(int argc,char *argv[])
{
	int clsoc1;
	struct sockaddr_in claddr1; char str[50];
	clsoc1=socket(AF_INET,SOCK_STREAM,0);
```

```
        memset(&claddr1,'\0',sizeof(claddr1));
        if(clsoc1==-1)
        {
                printf("error in creation\n");
        }
        else
        {
                claddr1.sin_family=AF_INET;
                claddr1.sin_port=htons(4955);
                claddr1.sin_addr.s_addr=inet_addr("127.0.0.1");
                int conn_stat=connect(clsoc1,(struct sockaddr*)&claddr1,sizeof(claddr1));
                if(conn_stat==-1)
                {
                        printf("Connection error\n");
                }
                else
                {
                        int recvid=recv(clsoc1,str,sizeof(str),0);
                        if(recvid<0)
                        {
                                printf("Send unsuccessful");
                        }
                        printf("Reversed String: %s\n",str);
                }
        }
        return 0;
}
```

OUTPUT

```
root@DESKTOP-70V9DEP:~# gcc server5a.c
root@DESKTOP-70V9DEP:~# ./a.out
SUCCESSFULLY RECEIVED
SENDING SUCCESSFUL

root@DESKTOP-70V9DEP:~# gcc client15a.c
root@DESKTOP-70V9DEP:~# ./a.out
Enter String: computer networks

root@DESKTOP-70V9DEP:~# gcc client25a.c
root@DESKTOP-70V9DEP:~# ./a.out
Reversed String: skrowten retupmoc
```

IMPLEMENTATION
B)

```
/* NAME : BHAGYA A JAI
   ROLL NO : B21CSB18
   EXPERIMENT : SQUARING A NUMBER*/
```

SERVER

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<string.h>
#include<arpa/inet.h>
int main()
{
        int sesoc;
        struct sockaddr_in serveraddr;
        int newsoc;
        struct sockaddr_in newaddr; socklen_t addr_size;
        int x;
        sesoc=socket(AF_INET,SOCK_STREAM,0);
        memset(&serveraddr,'\0',sizeof(serveraddr));
        if(sesoc==-1)
        {
                printf("socket creation failed");
        }
        else
        {
                serveraddr.sin_family=AF_INET;
                serveraddr.sin_port=htons(4955);
                serveraddr.sin_addr.s_addr=inet_addr("127.0.0.1");
                bind(sesoc,(struct sockaddr *)&serveraddr,
                sizeof(serveraddr));
                listen(sesoc,5); addr_size=sizeof(newsoc);
                newsoc=accept(sesoc,(struct sockaddr *)&newaddr,&addr_size);
                recv(newsoc,&x,sizeof(int),0); printf("SUCCESSFULLY RECEIVED\n");
                x=x*x;
                int newsoc1=accept(sesoc,(struct sockaddr*)&newaddr,&addr_size);
                int sid=send(newsoc1,&x,sizeof(int),0);
                if(sid<0)
                {
                        printf("Reverse Unsuccessful\n");
                }
                printf("SENDING SUCCESSFUL\n");
        }
        return 0;
}
```

CLIENT1

```
#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>
int main(int argc,char *argv[])
{
	int clsoc;
	struct sockaddr_in claddr; int x;
	clsoc=socket(AF_INET,SOCK_STREAM,0);
	memset(&claddr,'\0',sizeof(claddr));
	if(clsoc==-1)
	{
		printf("error in creation\n");
	}
	else
	{
		claddr.sin_family=AF_INET;
		claddr.sin_port=htons(4955);
		claddr.sin_addr.s_addr=inet_addr("127.0.0.1");
		int conn_stat=connect(clsoc,(struct sockaddr*)&claddr,sizeof(claddr));
		if(conn_stat==-1)
		{
			printf("Connection error\n");
		}
		else
		{
			printf("Enter Value: ");
			scanf("%d",&x);
			send(clsoc,&x,sizeof(int),0);
		}
	}
	return 0;
}
```

CLIENT2

```
#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>
int main(int argc,char *argv[])
{
	int clsoc1;
	struct sockaddr_in claddr1; int x;
	clsoc1=socket(AF_INET,SOCK_STREAM,0);
	memset(&claddr1,'\0',sizeof(claddr1));
	if(clsoc1==-1)
	{
		printf("error in creation\n");
	}
	else
	{
```

```
                claddr1.sin_family=AF_INET;
                claddr1.sin_port=htons(4955);
                claddr1.sin_addr.s_addr=inet_addr("127.0.0.1");
                int conn_stat=connect(clsoc1,(struct sockaddr*)&claddr1,sizeof(claddr1));
                //printf("%s",str);
                if(conn_stat==-1)
                {
                        printf("Connection error\n");
                }
                else
                {
                        int recvid=recv(clsoc1,&x,sizeof(int),0);
                        if(recvid<0)
                        {
                                printf("Send unsuccessful");
                        }
                        printf("Result: %d\n",x);
                }
        }
        return 0;
}
```

OUTPUT

```
root@DESKTOP-70V9DEP:~# gcc server5b.c
root@DESKTOP-70V9DEP:~# ./a.out
SUCCESSFULLY RECEIVED
SENDING SUCCESSFUL

root@DESKTOP-70V9DEP:~# gcc client15b.c
root@DESKTOP-70V9DEP:~# ./a.out
Enter Value: 6

root@DESKTOP-70V9DEP:~# gcc client25b.c
root@DESKTOP-70V9DEP:~# ./a.out
Result: 36
```

IMPLEMENTATION

C)

```
/* NAME : BHAGYA A JAI
   ROLL NO : B21CSB18
   EXPERIMENT : POWER 1.5*/
```

SERVER

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<string.h>
#include<arpa/inet.h>
#include<math.h>
int main()
{
        int sesoc;
        struct sockaddr_in serveraddr; int newsoc;
        struct sockaddr_in newaddr;
        socklen_t addr_size; float f;
        sesoc=socket(AF_INET,SOCK_STREAM,0);
        memset(&serveraddr,'\0',sizeof(serveraddr));
        if(sesoc==-1)
        {
                printf("socket creation failed");
        }
        else
        {
                serveraddr.sin_family=AF_INET;
                serveraddr.sin_port=htons(4955);
                serveraddr.sin_addr.s_addr=inet_addr("127.0.0.1");
                bind(sesoc,(struct sockaddr *)&serveraddr,sizeof(serveraddr));
                listen(sesoc,5); addr_size=sizeof(newsoc);
                newsoc=accept(sesoc,(struct sockaddr *)&newaddr,&addr_size);
                recv(newsoc,&f,sizeof(float),0); printf("SUCCESSFULLY RECEIVED \n");
                f=pow(f,1.5);
                newsoc=accept(sesoc,(struct sockaddr *)&newaddr,&addr_size);
                int sid=send(newsoc,&f,sizeof(float),0);
                if(sid<0)
                {
                        printf("Reverse Unsuccessful\n");
                }
                printf("SENDING SUCCESSFUL\n");
        }
        return 0;
}
```

CLIENT1

```
#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>
int main(int argc,char *argv[])
{
	int clsoc; float avg;
	struct sockaddr_in claddr;
	float f;
	clsoc=socket(AF_INET,SOCK_STREAM,0);
	memset(&claddr,'\0',sizeof(claddr));
	if(clsoc==-1)
	{
		printf("error in creation\n");
	}
	else
	{
		claddr.sin_family=AF_INET;
		claddr.sin_port=htons(4955);
		claddr.sin_addr.s_addr=inet_addr("127.0.0.1");
		int conn_stat=connect(clsoc,(struct sockaddr*)&claddr,sizeof(claddr));
		if(conn_stat==-1)
		{
			printf("Connection error\n");
		}
		else
		{
			printf("Enter Value: ");
			scanf("%f",&f);
			send(clsoc,&f,sizeof(float),0);
		}
		return 0;
	}
}
```

CLIENT2

```
#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>
int main(int argc,char *argv[])
{
	int clsoc1;
	struct sockaddr_in claddr1; float f;
	clsoc1=socket(AF_INET,SOCK_STREAM,0);
	memset(&claddr1,'\0',sizeof(claddr1)); if(clsoc1==-1)
	{
		printf("error in creation\n");
	}
	else
	{
		claddr1.sin_family=AF_INET;
```

```
                claddr1.sin_port=htons(4955);
                claddr1.sin_addr.s_addr=inet_addr("127.0.0.1");
                int conn_stat=connect(clsoc1,(struct sockaddr*)&claddr1,sizeof(claddr1));
                if(conn_stat==-1)
                {
                        printf("Connection error\n");
                }
                else
                {
                        int recvid=recv(clsoc1,&f,sizeof(float),0);
                        if(recvid<0)
                        {
                                printf("Send unsuccessful");
                        }
                        printf("Result: %f\n",f);
                }
        }
        return 0;
}
```

OUTPUT

```
root@DESKTOP-70V9DEP:~# gcc server5c.c -lm -o server5c
root@DESKTOP-70V9DEP:~# ./server5c
SUCCESSFULLY RECEIVED
SENDING SUCCESSFUL

root@DESKTOP-70V9DEP:~# gcc client15c.c
root@DESKTOP-70V9DEP:~# ./a.out
Enter Value: 5

root@DESKTOP-70V9DEP:~# gcc client25c.c
root@DESKTOP-70V9DEP:~# ./a.out
Result: 11.180340
```

IMPLEMENTATION

```
/* NAME : BHAGYA A JAI
   ROLL NO : B21CSB18
   EXPERIMENT : MULTIUSER CHAT SERVER*/
```

SERVER

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <string.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
   char message[50];
   int sockfd, client;
   sockfd = socket(AF_INET, SOCK_STREAM, 0);
   if (sockfd == -1)
   {
      printf("Socket creation failed!\n");
      return -1;
   }
   printf("Socket creation success\n");
   struct sockaddr_in server;
   server.sin_addr.s_addr = inet_addr("127.0.0.1");
   server.sin_family = AF_INET;
   server.sin_port = htons(4444);
   int size = sizeof(server);
   if (bind(sockfd, (struct sockaddr*)&server, sizeof(server)) < 0)
   {
      printf("Binding failed!\n");
      return -1;
   }
   printf("Binding successful\n");
   if (listen(sockfd, 1) < 0)
   {
      printf("Listening failed\n");
      return -1;
   }
   printf("Listening...\n");
   while (1)
   {
      client = accept(sockfd, (struct sockaddr*)&server, (socklen_t*)&size);
      if (client < 0)
      {
         printf("Acceptance error!\n");
         return -1;
      }
      printf("Client accepted\n");
```

```
        while (1)
        {
            recv(client, message, sizeof(message), 0);
            if (strcmp(message, "stop") == 0)
            {
                printf("Disconnected from client\n");
                break;
            }
            else
            {
                printf("Client : %s\n", message);
                printf("Server : ");
                scanf(" %[^\n]s", message);
                send(client, message, sizeof(message), 0);
            }
        }
    }
    close(client);
    return 0;
}
```

CLIENT

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <string.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <unistd.h>
#define LIMIT 25
int main()
{
    char message[50];
    int wordCount = 0;
    int sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd == -1)
    {
        printf("Socket creation failed!\n");
        return -1;
    }
    printf("Socket creation success\n");
    struct sockaddr_in server;
    server.sin_addr.s_addr = inet_addr("127.0.0.1");
    server.sin_family = AF_INET;
    server.sin_port = htons(4444);
    if (connect(sockfd, (struct sockaddr *)&server, sizeof(server)) < 0)
    {
        printf("Connection error\n");
        return -1;
    }
    printf("Connected to server\n");
```

```
    while (1)
    {
        printf("Client : ");
        scanf(" %[^\n]s", message);
        for (int i = 0; message[i] != '\0'; i++)
        {
            if (message[i] == ' ' || message[i] == '\n' || message[i] == '\t')
                wordCount++;
        }
        wordCount++;
        if (wordCount > LIMIT)
        {
            printf("You exceeded your daily limit\nDisconnected from server\n");
            strcpy(message, "stop");
            send(sockfd, message, sizeof(message), 0);
            close(sockfd);
            break;
        }
        send(sockfd, message, sizeof(message), 0);
        if (strcmp(message, "stop") == 0)
        {
            close(sockfd);
            printf("Disconnected from server\n");
            break;
        }
        recv(sockfd, message, sizeof(message), 0);
        printf("Server : %s\n", message);
    }
    return 0;
}
```

OUTPUT

```
root@DESKTOP-70V9DEP:~# gcc server6.c
root@DESKTOP-70V9DEP:~# ./a.out
Socket creation success
Binding successful
Listening...
Client accepted
Client : hello server
Server : hi client
Client : cn i get a help from you?
Server : sure,tell me
Client : can i know more about sockets?
Server : let's have a meeting on that.
Client : stop.
Server : okay
Disconnected from client

root@DESKTOP-70V9DEP:~# gcc client6.c
root@DESKTOP-70V9DEP:~# ./a.out
Socket creation success
Connected to server
```

Client : hello server
Server : hi client
Client : cn i get a help from you?
Server : sure,tell me
Client : can i know more about sockets?
Server : let's have a meeting on that.
Client : stop.
Server : okay
Client : stop
Disconnected from server

IMPLEMENTATION

```
/* NAME : BHAGYA A JAI
   ROLL NO : B21CSB18
   EXPERIMENT : TIME SERVER APPLICATION*/
```

SERVER

```
#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 <time.h>
#define MAXLINE 1024
int main() {
   int sockfd;
   char buffer[MAXLINE];
   time_t t;
   time(&t);
   char systime[50];
   strcpy(systime, ctime(&t));
   if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
      printf("Socket creation failed\n");
      return -1;
   }
   printf("Socket creation success\n");
   struct sockaddr_in server, client;
   server.sin_family = AF_INET;
   server.sin_addr.s_addr = inet_addr("127.0.0.1");
   server.sin_port = htons(2500);
   int size = sizeof(server);
   if (bind(sockfd, (const struct sockaddr *)&server, size) < 0) {
      printf("Binding failed\n");
      return -1;
   }
   printf("Binding success\n");
   int n = recvfrom(sockfd, (char *)buffer, MAXLINE, MSG_WAITALL, (struct sockaddr
*)&server, &size);
   buffer[n] = '\0';
   printf("Client time : %s", buffer);
   sendto(sockfd, (const char *)systime, sizeof(systime), 0, (const struct sockaddr *)&server, size);
   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 <time.h>
```

```
#define MAXLINE 1024
int main() {
   int sockfd, n, len;
   char buffer[MAXLINE];
   time_t t;
   time(&t);
   char systime[50];
   strcpy(systime, ctime(&t));
   if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
      printf("Socket creation failed\n");
      return -1;
   }
   printf("Socket creation success\n");
   struct sockaddr_in server;
   server.sin_addr.s_addr = inet_addr("127.0.0.1");
   server.sin_family = AF_INET;
   server.sin_port = htons(2500);
   int size = sizeof(server);
   sendto(sockfd, (const char *)systime, sizeof(systime), 0, (const struct sockaddr *) &server, size);
   n = recvfrom(sockfd, buffer, MAXLINE, 0, (struct sockaddr *)&server, (socklen_t*)&size);
   buffer[n] = '\0';
   printf("Server time : %s", buffer);
   close(sockfd);
   return 0;
}
```

OUTPUT

```
root@DESKTOP-70V9DEP:~# gcc timeserver.c
root@DESKTOP-70V9DEP:~# ./a.out
Socket creation success
Binding success
Client time : Fri Jun 28 20:00:35 2024
root@DESKTOP-70V9DEP:~# gcc timeclient.c
root@DESKTOP-70V9DEP:~# ./a.out
Socket creation success
Server time : Fri Jun 28 20:00:24 2024
```

IMPLEMENTATION

A)

```
/* NAME : BHAGYA A JAI
   ROLL NO : B21CSB18
   EXPERIMENT : STOP AND WAIT ARQ*/
```

SERVER

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <pthread.h>
#define PORT 8000
#define SIZE 100
typedef struct packet {
    int data;
    int type; // SEQ (0) or ACK (1)
    int seq; // Sequence number (0 or 1)
} packet;
void main() {
    int server_fd, client_fd;
    struct sockaddr_in address;
    int addrlen = sizeof(address);
    int arr[SIZE], k = 0;
    for(int i = 0; i < SIZE; i++)
        arr[i] = -1;
    printf("Stop and Wait ARQ\nTCP Server\n");
    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        printf("Socket creation failed!\n");
        exit(1);
    }
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(PORT);
    if (bind(server_fd, (struct sockaddr*) &address, addrlen) < 0) {
        printf("Socket binding failed!\n");
        exit(1);
    }
    if (listen(server_fd, 5) < 0) {
        printf("Listening failed!\n");
        exit(1);
    }
    if ((client_fd = accept(server_fd, (struct sockaddr*) &address, (socklen_t*) &addrlen)) < 0) {
        printf("Connection failed!\n");
        exit(1);
    } else {
        printf("Connected to client.\n");
    }
    packet p;
    int flag = -1;
    while (1) {
        int status = recv(client_fd, &p, sizeof(packet), 0);
```

```
        if (status < 0) {
            printf("Receive failed!\n");
        } else if (status == 0) {
            printf("Receive completed.\nArray: ");
            for (int i = 0; arr[i] != -1; i++) {
                printf("%d ", arr[i]);
            }
            printf("\n");
        } else {
            if (flag != p.seq) {
                arr[k] = p.data;
                k++;
            }
            printf("Received: %d (SEQ %d)\n", p.data, p.seq);
            flag = p.seq;
            p.type = 1;
            p.seq = (p.seq + 1) % 2;
            if (rand() % 5 != 2) {
                if (send(client_fd, &p, sizeof(packet), 0) < 0) {
                    printf("Send failed!\n");
                } else {
                    printf("Sent: ACK %d\n", p.seq);
                }
            } else {
                printf("ACK %d lost\n", p.seq);
            }
        }
    }
    close(server_fd);
    close(client_fd);
}
```

CLIENT

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <pthread.h> //  gcc stopwaitc.c -lpthread -o stopwaitc – to compile
#define PORT 8000
typedef struct packet {
    int data;
    int type; // SEQ (0) or ACK (1)
    int seq; // Sequence number (0 or 1)
} packet;
typedef struct data {
    int* arr;
    int* i;
    int client_fd;
    packet* p;
} data;
void* client(void* arg) {
```

```
    data d = *((data*) arg);
    d.p->type = 0;
    d.p->data = d.arr[*d.i];
    if (rand() % 5 != 2) {
        if (send(d.client_fd, d.p, sizeof(packet), 0) < 0) {
            printf("Send failed!\n");
        } else {
            printf("Sent: %d (SEQ %d)\n", d.p->data, d.p->seq);
            if (recv(d.client_fd, d.p, sizeof(packet), 0) < 0) {
                printf("Receive failed!\n");
            } else {
                printf("Received: ACK %d\n", d.p->seq);
                d.arr[*d.i] = -1;
                *(d.i) = *(d.i) + 1;
            }
        }
    } else {
        printf("SEQ %d lost\n", d.p->seq);
    }
}
void* timeout(void* t) {
    sleep(1);
    pthread_t tid = *((pthread_t*) t);
    pthread_cancel(tid);
}
void main() {
    int client_fd;
    struct sockaddr_in serv_addr;
    printf("TCP Client\n");
    client_fd = socket(AF_INET, SOCK_STREAM, 0);
    if (client_fd < 0) {
        printf("Socket creation failed!\n");
        exit(1);
    }
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = INADDR_ANY;
    serv_addr.sin_port = htons(PORT);
    if (connect(client_fd, (struct sockaddr*) &serv_addr, sizeof(serv_addr)) < 0) {
        printf("Connection failed!\n");
        exit(1);
    }
    int n;
    printf("Connected to server.\n");
    printf("Enter array size: ");
    scanf("%d", &n);
    int arr[n];
    printf("Enter array elements: ");
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
    int i = 0;
    packet p;
```

```
    data d;
    d.client_fd = client_fd;
    d.p = &p;
    d.arr = arr;
    d.i = &i;
    p.seq = 0;
    pthread_t tid1, tid2;
    while (1) {
        if (i == n) {
            printf("Send completed.\nArray: ");
            for (int j = 0; j < n; j++) {
                printf("%d ", arr[j]);
            }
            printf("\n");
            break;
        }
        pthread_create(&tid1, NULL, client, &d);
        pthread_create(&tid2, NULL, timeout, &tid1);
        pthread_join(tid1, NULL);
        pthread_join(tid2, NULL);
    }
    close(client_fd);
}
```

OUTPUT

Server

```
root@DESKTOP-70V9DEP:~# gcc stopwaits.c -o stopwaits -pthread
root@DESKTOP-70V9DEP:~# ./stopwaits
Stop and Walt ARQ
TCP Server
Connected to client. Received: 1 (SEQ 0)
Sent: ACK 1
Received: 2 (SEQ 1) Sent: ACK O
Received: 3 (5EQ 0)
ACK 1 lost
Received: 3 (SEQ 0) Sent: ACK 1
Received: 4 (SEQ 1)
Sent: ACK O Received: 5 (SEQ 0)
Sent: ACK 1
Receive completed. Array: 1 2 3 4 5
```

Client

```
root@DESKTOP-70V9DEP:~# gcc stopwaitc.c -o stopwaitc -pthread
root@DESKTOP-70V9DEP:~# ./stopwaitc
TCP Client
Connected to server.
Enter array size: 5
Enter array elements: 1 2 3 4 5
Sent: 1 (SEQ 0)
Received: ACK 1
Sent: 2 (SEQ 1)
Received: ACK 0
SEQ 0 lost
```

Sent: 3 (SEQ 0)
Sent: 3 (SEQ 0)
Received: ACK 1
Sent: 4 (SEQ 1)
Received: ACK 0
Sent: 5 (SEQ 0)
Received: ACK 1
Send completed.
Array: -1 -1 -1 -1 -1

IMPLEMENTATION
B)

```
/* NAME : BHAGYA A JAI
   ROLL NO : B21CSB18
   EXPERIMENT : GO BACK N ARQ*/
```

SERVER

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#define PORT 8004
#define SIZE 100
typedef struct packet {
    int data;
    int type; // SEQ (0), ACK (1) or NACK (-1)
    int seq;  // Sequence number
} packet;
void main() {
    int server_fd, client_fd;
    struct sockaddr_in address;
    int addrlen = sizeof(address);
    int arr[SIZE];
    for (int i = 0; i < SIZE; i++) {
        arr[i] = -1;
    }
    printf("Go-Back-N ARQ\nTCP Server\n");
    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        printf("Socket creation failed!\n");
        exit(1);
    }
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(PORT);
    if (bind(server_fd, (struct sockaddr*)&address, addrlen) < 0) {
        printf("Socket binding failed!\n");
        exit(1);
    }
    if (listen(server_fd, 5) < 0) {
        printf("Listening failed!\n");
        exit(1);
    }
    if ((client_fd = accept(server_fd, (struct sockaddr*)&address, (socklen_t*)&addrlen)) < 0) {
        printf("Connection failed!\n");
        exit(1);
    } else {
        printf("Connected to client.\n");
    }
    packet p;
    int exp_seq = 0, flag = 0;
    while (1) {
        int status = recv(client_fd, &p, sizeof(packet), 0);
```

```
        if (status < 0) {
            printf("Receive failed!\n");
        } else if (status == 0) {
            // End of transmission
            printf("Receive completed.\nArray: ");
            for (int i = 0; arr[i] != -1; i++) {
                printf("%d ", arr[i]);
            }
            printf("\n");
            break;
        } else {
            if (p.seq > exp_seq) {
                // Out of order packet received, send NACK
                if (!flag) {
                    flag = 1;
                    p.type = -1;
                    p.seq = exp_seq;
                    if (send(client_fd, &p, sizeof(packet), 0) < 0) {
                        printf("Send failed!\n");
                    } else {
                        printf("Sent: NACK %d\n", p.seq);
                    }
                }
            } else {
                // In-order packet received, send ACK
                flag = 0;
                exp_seq = p.seq + 1;
                p.type = 1;
                printf("Received: %d (SEQ %d)\n", p.data, p.seq);
                arr[p.seq] = p.data;
                if (rand() % 10 != 6) {
                    if (send(client_fd, &p, sizeof(packet), 0) < 0) {
                        printf("Send failed!\n");
                    } else {
                        printf("Sent: ACK %d\n", p.seq);
                    }
                } else {
                    printf("ACK %d lost\n", p.seq);
                }
            }
        }
    }
    close(server_fd);
    close(client_fd);
}
```

CLIENT

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
```

```
#define PORT 8004
typedef struct packet {
    int data;
    int type; // SEQ (0), ACK (1) or NACK(-1)
    int seq;  // Sequence number
} packet;
typedef struct window {
    int size;
    int start;
    int end;
} window;
typedef struct data {
    int* arr;
    int n;
    int client_fd;
    int exp_seq;
    packet* p;
    window* w;
} data;
void recvAck(data d);
void sendWindow(data d) {
    d.p->seq = d.w->start;
    for(int i = d.w->start; i <= d.w->end && i < d.n; i++) {
        d.p->type = 0;
        d.p->data = d.arr[i];
        if(rand() % 10 != 6) {
            if(send(d.client_fd, d.p, sizeof(packet), 0) < 0) {
                printf("Send failed!\n");
            } else {
                printf("Sent: %d (SEQ %d)\n", d.p->data, d.p->seq);
            }
        } else {
            printf("%d (SEQ %d) lost\n", d.p->data, d.p->seq);
        }
        d.p->seq = d.p->seq + 1;
    }
    recvAck(d);
}
void sendFrame(data d) {
    d.p->type = 0;
    d.p->data = d.arr[d.w->end];
    if(rand() % 10 != 6) {
        if(send(d.client_fd, d.p, sizeof(packet), 0) < 0) {
            printf("Send failed!\n");
        } else {
            printf("Sent: %d (SEQ %d)\n", d.p->data, d.p->seq);
        }
    } else {
        printf("%d (SEQ %d) lost\n", d.p->data, d.p->seq);
    }
    d.p->seq = d.p->seq + 1;
```

```
    recvAck(d);
}
void recvAck(data d) {
    data d1;
    packet p;
    d1.p = &p;
    if(recv(d.client_fd, d1.p, sizeof(packet), 0) < 0) {
        printf("Time out! Window retransmitting.\n");
        sendWindow(d);
    } else {
        if(d1.p->seq > d.exp_seq) {
            printf("ACK %d not received! Window retransmitting.\n", d.exp_seq);
            while(recv(d.client_fd, d1.p, sizeof(packet), 0) > 0);
            sendWindow(d);
            return;
        }
        if(d1.p->type == 1) {
            printf("Received: ACK %d\n", d1.p->seq);
            d.arr[d1.p->seq] = -1;
            d.w->start++;
            if(d.w->start == d.n) {
                printf("Send completed.\nArray: ");
                for(int i = 0; i < d.n; i++) {
                    printf("%d ", d.arr[i]);
                }
                printf("\n");
                close(d.client_fd);
                exit(0);
            }
            d.w->end++;
            d.exp_seq = d1.p->seq + 1;
            if(d.w->end < d.n)
                sendFrame(d);
            else
                recvAck(d);
        } else if(d1.p->type == -1) {
            printf("Received: NACK %d. Window retransmitting.\n", d1.p->seq);
            sendWindow(d);
        }
    }
}
void main() {
    int client_fd;
    struct sockaddr_in serv_addr;
    printf("TCP Client\n");
    client_fd = socket(AF_INET, SOCK_STREAM, 0);
    if(client_fd < 0) {
        printf("Socket creation failed!\n");
        exit(1);
    }
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = INADDR_ANY;
```

```
    serv_addr.sin_port = htons(PORT);
    if(connect(client_fd, (struct sockaddr*) &serv_addr, sizeof(serv_addr)) < 0) {
        printf("Connection failed!\n");
        exit(1);
    } else {
        printf("Connected to server.\n");
    }
    struct timeval tv;
    tv.tv_sec = 1;
    tv.tv_usec = 0;
    setsockopt(client_fd, SOL_SOCKET, SO_RCVTIMEO, (const char*)&tv, sizeof tv);
    int n;
    window w;
    printf("Enter window size: ");
    scanf("%d", &w.size);
    w.start = 0;
    w.end = w.size - 1;
    printf("Enter array size: ");
    scanf("%d", &n);
    int arr[n];
    printf("Enter array elements: ");
    for(int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
    packet p;
    data d;
    d.client_fd = client_fd;
    d.p = &p;
    d.w = &w;
    d.n = n;
    d.arr = arr;
    d.exp_seq = 0;
    p.seq = 0;
    sendWindow(d);
}
```

OUTPUT

```
root@DESKTOP-70V9DEP:~# gcc gobacks.c -o gobacks -pthread
root@DESKTOP-70V9DEP:~# ./gobacks
Go-Back-N ARQ
TCP Server
Connected to client.
Received: 1 (SEQ 0)
Sent: ACK 0
Sent: NACK 1
Received: 2 (SEQ 1)
ACK 1 lost
Received: 3 (SEQ 2)
Sent: ACK 2
Received: 2 (SEQ 1)
Sent: ACK 1
Received: 3 (SEQ 2)
Sent: ACK 2
```

Received: 4 (SEQ 3)
Sent: ACK 3
Received: 5 (SEQ 4)
ACK 4 lost
Received: 6 (SEQ 5)
Sent: ACK 5
Received: 7 (SEQ 6)
Sent: ACK 6
Received: 5 (SEQ 4)
Sent: ACK 4
Received: 6 (SEQ 5)
Sent: ACK 5
Sent: NACK 6
Received: 7 (SEQ 6)
Sent: ACK 6
Sent: NACK 7
Received: 8 (SEQ 7)
Sent: ACK 7
Received: 9 (SEQ 8)
Sent: ACK 8
Received: 10 (SEQ 9)
Sent: ACK 9
Receive completed.
Array: 1 2 3 4 5 6 7 8 9 10

root@DESKTOP-70V9DEP:~# gcc gobackc.c -o gobackc -pthread
root@DESKTOP-70V9DEP:~# ./gobackc
TCP Client
Connected to server.
Enter window size: 3
Enter array size: 10
Enter array elements: 1 2 3 4 5 6 7 8 9 10
Sent: 1 (SEQ 0)
2 (SEQ 1) lost
Sent: 3 (SEQ 2)
Received: ACK 0
Sent: 4 (SEQ 3)
Received: NACK 1. Window retransmitting.
Sent: 2 (SEQ 1)
Sent: 3 (SEQ 2)
4 (SEQ 3) lost
ACK 1 not received! Window retransmitting.
Sent: 2 (SEQ 1)
Sent: 3 (SEQ 2)
Sent: 4 (SEQ 3)
Received: ACK 1
Sent: 5 (SEQ 4)
Received: ACK 2
Sent: 6 (SEQ 5)
Received: ACK 3
Sent: 7 (SEQ 6)
ACK 4 not received! Window retransmitting.

Sent: 5 (SEQ 4)
Sent: 6 (SEQ 5)
7 (SEQ 6) lost
Received: ACK 4
Sent: 8 (SEQ 7)
Received: ACK 5
9 (SEQ 8) lost
Received: NACK 6. Window retransmitting.
Sent: 7 (SEQ 6)
8 (SEQ 7) lost
Sent: 9 (SEQ 8)
Received: ACK 6
Sent: 10 (SEQ 9)
Received: NACK 7. Window retransmitting.
Sent: 8 (SEQ 7)
Sent: 9 (SEQ 8)
Sent: 10 (SEQ 9)
Received: ACK 7
Received: ACK 8
Received: ACK 9
Send completed.
Array: -1 -1 -1 -1 -1 -1 -1 -1 -1 -1

IMPLEMENTATION
C)

```
/* NAME : BHAGYA A JAI
   ROLL NO : B21CSB18
   EXPERIMENT : SELECTIVE REPEAT ARQ*/
```

SERVER

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <pthread.h>
#define PORT 8005
#define SIZE 100
typedef struct packet {
   int data;
   int type; // SEQ (0), ACK (1) or NACK(-1)
   int seq; // Sequence number
} packet;
int add(int* arr, int key, int index) {
   int flag = -1;
   for(int i = 0; i < index; i++) {
      if(arr[i] == -1) {
         flag = i;
         break;
      }
   }
   arr[index] = key;
   return flag;
}
void main() {
   int server_fd, client_fd;
   struct sockaddr_in address;
   int addrlen = sizeof(address);
   printf("Selective Repeat ARQ\nTCP Server\n");
   if((server_fd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
      printf("Socket creation failed!\n");
      exit(1);
   }
   address.sin_family = AF_INET;
   address.sin_addr.s_addr = INADDR_ANY;
   address.sin_port = htons(PORT);
   if(bind(server_fd, (struct sockaddr*) &address, addrlen) < 0) {
      printf("Socket binding failed!\n");
      exit(1);
   }
   if(listen(server_fd, 5) < 0) {
      printf("Listening failed!\n");
      exit(1);
   }
   if((client_fd = accept(server_fd, (struct sockaddr*) &address, (socklen_t*) &addrlen)) < 0) {
      printf("Connection failed!\n");
```

```
        exit(1);
    } else {
        printf("Connected to client.\n");
    }
    packet p;
    int* arr = malloc(SIZE * sizeof(int));
    for(int i = 0; i < SIZE; i++)
        arr[i] = -1;
    while(1) {
        int status = recv(client_fd, &p, sizeof(packet), 0);
        if(status < 0) {
            printf("Receive failed!\n");
        } else if (status == 0) {
            printf("Receive completed.\nArray: ");
            for(int i = 0; arr[i] != -1; i++) {
                printf("%d ", arr[i]);
            }
            printf("\n");
            break;
        } else {
            printf("Received: %d (SEQ %d)\n", p.data, p.seq);
            int index = add(arr, p.data, p.seq);
            if(index != -1) {
                int temp = p.seq;
                p.type = -1;
                p.seq = index;
                if(rand() % 10 != 6) {
                    if(send(client_fd, &p, sizeof(packet), 0) < 0) {
                        printf("Send failed!\n");
                    } else {
                        printf("Sent: NACK %d\n", p.seq);
                    }
                } else {
                    printf("Lost: NACK %d\n", p.seq);
                }
                p.seq = temp;
            }
            p.type = 1;
            if(rand() % 10 != 6) {
                if(send(client_fd, &p, sizeof(packet), 0) < 0) {
                    printf("Send failed!\n");
                } else {
                    printf("Sent: ACK %d\n", p.seq);
                }
            } else {
                printf("Lost: ACK %d\n", p.seq);
            }
        }
    }
    close(server_fd);
    close(client_fd);
}
```

CLIENT

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#define PORT 8005
int count = 0;
typedef struct packet {
    int data;
    int type; // SEQ (0), ACK (1) or NACK(-1)
    int seq; // Sequence number
} packet;
typedef struct window {
    int size;
    int start;
    int end;
} window;
typedef struct data {
    int* arr;
    int n;
    int client_fd;
    packet* p;
    window* w;
} data;
int ackFrame(int* arr, int index) {
    int flag = -1;
    for(int i = 0; i < index; i++) {
        if(arr[i] != -1) {
            flag = i;
            break;
        }
    }
    arr[index] = -1;
    return flag;
}
void sendWindow(data d) {
    for(d.p->seq = d.w->start; d.p->seq <= d.w->end && d.p->seq < d.n; d.p->seq++) {
        d.p->type = 0;
        d.p->data = d.arr[d.p->seq];
        if(d.p->data == -1)
            continue;
        if(rand() % 10 != 6) {
            if(send(d.client_fd, d.p, sizeof(packet), 0) < 0) {
                printf("Send failed!\n");
            } else {
                printf("Sent: %d (SEQ %d)\n", d.p->data, d.p->seq);
            }
        } else {
            printf("Lost: %d (SEQ %d)\n", d.p->data, d.p->seq);
        }
    }
```

```
}
void sendFrame(data d, int seq) {
    d.p->type = 0;
    int temp;
    if(seq == -1)
        d.p->data = d.arr[d.w->end];
    else {
        d.p->data = d.arr[seq];
        temp = d.p->seq;
        d.p->seq = seq;
    }
    if(d.p->data == -1)
        return;
    if(rand() % 10 != 6) {
        if(send(d.client_fd, d.p, sizeof(packet), 0) < 0) {
            printf("Send failed!\n");
        } else {
            printf("Sent: %d (SEQ %d)\n", d.p->data, d.p->seq);
        }
    } else {
        printf("Lost: %d (SEQ %d)\n", d.p->data, d.p->seq);
    }
    if(seq == -1)
        d.p->seq = d.p->seq + 1;
    else
        d.p->seq = temp;
}
void recvAck(data d) {
    data d1;
    packet p;
    d1.p = &p;
    if(recv(d.client_fd, d1.p, sizeof(packet), 0) < 0) {
        printf("Time out! Window retransmitting.\n");
        sendWindow(d);
        recvAck(d);
    } else {
        if(d1.p->type == 1) {
            if(d.arr[d1.p->seq] == -1) {
                recvAck(d);
            } else {
                printf("Received: ACK %d\n", d1.p->seq);
                count++;
                d.w->start++;
                d.w->end++;
                int index = ackFrame(d.arr, d1.p->seq);
                if(index != -1) {
                    printf("ACK %d not received! Frame %d retransmitting.\n", index, index);
                    sendFrame(d, index);
                }
                if(count == d.n) {
                    printf("Send completed.\nArray: ");
                    for(int i = 0; i < d.n; i++) {
```

```
                printf("%d ", d.arr[i]);
            }
            printf("\n");
            close(d.client_fd);
            exit(0);
        }
        if(d.w->end < d.n) {
            sendFrame(d, -1);
            recvAck(d);
        } else {
            recvAck(d);
        }
      }
    } else if(d1.p->type == -1) {
      printf("Received: NACK %d. Frame %d retransmitting.\n", d1.p->seq, d1.p->seq);
      sendFrame(d, d1.p->seq);
      recvAck(d);
    }
  }
}
void main() {
  int client_fd;
  struct sockaddr_in serv_addr;
  printf("TCP Client\n");
  client_fd = socket(AF_INET, SOCK_STREAM, 0);
  if(client_fd < 0) {
    printf("Socket creation failed!\n");
    exit(1);
  }
  serv_addr.sin_family = AF_INET;
  serv_addr.sin_addr.s_addr = INADDR_ANY;
  serv_addr.sin_port = htons(PORT);
  if(connect(client_fd, (struct sockaddr*) &serv_addr, sizeof(serv_addr)) < 0) {
    printf("Connection failed!\n");
    exit(1);
  } else {
    printf("Connected to server.\n");
  }
  struct timeval tv;
  tv.tv_sec = 1;
  tv.tv_usec = 0;
  setsockopt(client_fd, SOL_SOCKET, SO_RCVTIMEO, (const char*)&tv, sizeof tv);
  int n;
  window w;
  printf("Enter window size: ");
  scanf("%d", &w.size);
  w.start = 0;
  w.end = w.size - 1;
  printf("Enter array size: ");
  scanf("%d", &n);
  int arr[n];
  printf("Enter array elements: ");
```

```
    for(int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
    packet p;
    data d;
    d.client_fd = client_fd;
    d.p = &p;
    d.w = &w;
    d.n = n;
    d.arr = arr;
    p.seq = 0;
    sendWindow(d);
    recvAck(d);
}
```

OUTPUT

```
root@DESKTOP-70V9DEP:~# gcc selreps.c -o selreps -pthread
root@DESKTOP-70V9DEP:~# ./selreps
Selective Repeat ARQ
TCP Server
Connected to client.
Received: 1 (SEQ 0)
Sent: ACK 0
Received: 3 (SEQ 2)
Lost: NACK 1
Sent: ACK 2
Received: 4 (SEQ 3)
Sent: NACK 1
Sent: ACK 3
Received: 2 (SEQ 1)
Sent: ACK 1
Received: 5 (SEQ 4)
Lost: ACK 4
Received: 2 (SEQ 1)
Sent: ACK 1
Received: 6 (SEQ 5)
Sent: ACK 5
Received: 7 (SEQ 6)
Sent: ACK 6
Received: 5 (SEQ 4)
Sent: ACK 4
Received: 8 (SEQ 7)
Sent: ACK 7
Received: 5 (SEQ 4)
Sent: ACK 4
Received: 9 (SEQ 8)
Sent: ACK 8
Received: 10 (SEQ 9)
Sent: ACK 9
Receive completed.
Array: 1 2 3 4 5 6 7 8 9 10

root@DESKTOP-70V9DEP:~# gcc selrepc.c -o selrepc -pthread
```

```
root@DESKTOP-70V9DEP:~# ./selrepc
TCP Client
Connected to server.
Enter window size: 3
Enter array size: 10
Enter array elements: 1 2 3 4 5 6 7 8 9 10
Sent: 1 (SEQ 0)
Lost: 2 (SEQ 1)
Sent: 3 (SEQ 2)
Received: ACK 0
Sent: 4 (SEQ 3)
Received: ACK 2
ACK 1 not received! Frame 1 retransmitting.
Sent: 2 (SEQ 1)
Sent: 5 (SEQ 4)
Received: NACK 1. Frame 1 retransmitting.
Lost: 2 (SEQ 1)
Received: ACK 3
ACK 1 not received! Frame 1 retransmitting.
Sent: 2 (SEQ 1)
Sent: 6 (SEQ 5)
Received: ACK 1
Sent: 7 (SEQ 6)
Received: ACK 5
ACK 4 not received! Frame 4 retransmitting.
Sent: 5 (SEQ 4)
Sent: 8 (SEQ 7)
Received: ACK 6
ACK 4 not received! Frame 4 retransmitting.
Sent: 5 (SEQ 4)
Sent: 9 (SEQ 8)
Received: ACK 4
Sent: 10 (SEQ 9)
Received: ACK 7
Received: ACK 8
Received: ACK 9
Send completed.
Array: -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
```

IMPLEMENTATION

```
/* NAME : BHAGYA A JAI
   ROLL NO : B21CSB18
   EXPERIMENT : DISTANCE VECTOR ROUTING*/

#include <stdio.h>
#include <limits.h>
struct node {
   int dist[100];
   int from[100];
};
int path(int src, int i, int from[]) {
   int temp;
   while (1) {
      temp = i;
      i = from[i];
      if (temp == i)
         break;
   }
   return temp;
}
void main() {
   int nodes;
   printf("Enter the number of nodes: ");
   scanf("%d", &nodes);
   int costmat[nodes][nodes];
   struct node rt[nodes];
   printf("\nEnter the cost matrix:\n");
   for (int i = 0; i < nodes; i++) {
      for (int j = 0; j < nodes; j++) {
         scanf("%d", &costmat[i][j]);
         if (costmat[i][j] == -1)
            costmat[i][j] = SHRT_MAX;
         rt[i].dist[j] = costmat[i][j];
         rt[i].from[j] = j;
      }
      costmat[i][i] = 0;
   }
   int count;
   do {
      count = 0;
      for (int i = 0; i < nodes; i++)
         for (int j = 0; j < nodes; j++)
            for (int k = 0; k < nodes; k++)
               if (rt[i].dist[j] > costmat[i][k] + rt[k].dist[j]) {
                  rt[i].dist[j] = rt[i].dist[k] + rt[k].dist[j];
                  rt[i].from[j] = k;
                  count++;
               }
   } while (count != 0);
   for (int i = 0; i < nodes; i++) {
      printf("\nFor router %d\n", i + 1);
```

```
        for (int j = 0; j < nodes; j++) {
            printf("\nShortest distance to router %d is %d via %d", j + 1,
                rt[i].dist[j], path(i, j, rt[i].from) + 1);
        }
        printf("\n");
    }
}
```

OUTPUT

```
root@DESKTOP-70V9DEP:~# gcc disvect.c
root@DESKTOP-70V9DEP:~# ./a.out
Enter the number of nodes: 4

Enter the cost matrix:
0 3 23 -1
3 0 2 -1
23 2 0 5 -1 -1 5 0

For router 1

Shortest distance to router 1 is 0 via 1
Shortest distance to router 2 is 3 via 2
Shortest distance to router 3 is 5 via 2
Shortest distance to router 4 is 10 via 2

For router 2

Shortest distance to router 1 is 3 via 1
Shortest distance to router 2 is 0 via 2
Shortest distance to router 3 is 2 via 3
Shortest distance to router 4 is 7 via 3

For router 3

Shortest distance to router 1 is 5 via 2
Shortest distance to router 2 is 2 via 2
Shortest distance to router 3 is 0 via 3
Shortest distance to router 4 is 5 via 4

For router 4

Shortest distance to router 1 is 10 via 3
Shortest distance to router 2 is 7 via 3
Shortest distance to router 3 is 5 via 3
Shortest distance to router 4 is 0 via 4
```

IMPLEMENTATION

```
/* NAME : BHAGYA A JAI
   ROLL NO : B21CSB18
   EXPERIMENT : LINK STATE ROUTING*/

#include <limits.h>
#include <stdio.h>
#include <stdbool.h>
int minDistance(int V, int dist[], bool sptSet[]) {
   int min = INT_MAX, min_index;
   for (int v = 0; v < V; v++)
      if (sptSet[v] == false && dist[v] <= min) {
         min = dist[v];
         min_index = v;
      }
   return min_index;
}
int path(int src, int i, int from[]) {
   int temp = from[i];
   if (temp == src) {
      return i;
   }
   i = from[i];
   while (i != src) {
      temp = i;
      i = from[i];
   }
   return temp;
}
void printSolution(int V, int dist[], int from[], int src) {
   printf("\nFor router %d\n", src + 1);
   printf("\nTo router\tShortest distance\tVia\n");
   for (int i = 0; i < V; i++)
      printf("%d\t\t%d\t\t\t%d\n", i + 1, dist[i], path(src, i, from) + 1);
}
void dijkstra(int V, int costmat[V][V], int src) {
   int dist[V];
   int from[V];
   bool sptSet[V];
   for (int i = 0; i < V; i++) {
      dist[i] = INT_MAX;
      from[i] = i;
      sptSet[i] = false;
   }
   dist[src] = 0;
   for (int count = 0; count < V - 1; count++) {
      int u = minDistance(V, dist, sptSet);
      sptSet[u] = true;
      for (int v = 0; v < V; v++)
         if (!sptSet[v] && costmat[u][v] && dist[u] != INT_MAX && dist[u] + costmat[u][v] <
dist[v]) {
            from[v] = u;
```

```
                dist[v] = dist[u] + costmat[u][v];
            }
        }
        printSolution(V, dist, from, src);
}
void main() {
    int nodes;
    printf("Enter the number of nodes: ");
    scanf("%d", &nodes);
    int costmat[nodes][nodes];
    printf("\nEnter the cost matrix:\n");
    for (int i = 0; i < nodes; i++) {
        for (int j = 0; j < nodes; j++) {
            scanf("%d", &costmat[i][j]);
            if (costmat[i][j] == -1)
                costmat[i][j] = SHRT_MAX;
        }
        costmat[i][i] = 0;
    }
    for (int i = 0; i < nodes; i++)
        dijkstra(nodes, costmat, i);
}
```

OUTPUT

```
root@DESKTOP-70V9DEP:~# gcc linkstate.c
root@DESKTOP-70V9DEP:~# ./a.out
Enter the number of nodes: 4

Enter the cost matrix:
0 3 23 -1
3 0 2 -1
23 2 0 5
-1 -1 5 0

For router 1

To router       Shortest distance       Via
1               0                       1
2               3                       2
3               5                       2
4               10                       2

For router 2

To router       Shortest distance       Via
1               3                       1
2               0                       2
3               2                       3
4               7                       3

For router 3

To router       Shortest distance       Via
```

1	5	2
2	2	2
3	0	3
4	5	4

For router 4

To router	Shortest distance	Via
1	10	3
2	7	3
3	5	3
4	0	4

IMPLEMENTATION

```
/* NAME : BHAGYA A JAI
   ROLL NO : B21CSB18
   EXPERIMENT : FILE TRANSFER PROTOCOL*/
```

SERVER

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <string.h>
#include <dirent.h>
void serverRecv(int client_fd) {
    char str[100];
    char str1[1000];
    if (recv(client_fd, str, 100 * sizeof(char), 0) <= 0) {
        printf("Connection lost!\n");
        return;
    }
    if (!strcmp(str, "GET")) {
        DIR *d;
        struct dirent *dir;
        d = opendir(".");
        str1[0] = '\0';
        if (d) {
            while ((dir = readdir(d)) != NULL) {
                if (strcmp(dir->d_name, ".") && strcmp(dir->d_name, "..") &&
                    strcmp(dir->d_name, "server.c") && strcmp(dir->d_name, "server")) {
                    strcat(str1, dir->d_name);
                    strcat(str1, "\n");
                }
            }
            closedir(d);
        }
        if (send(client_fd, str1, 1000 * sizeof(char), 0) <= 0) {
            printf("GET failed!\n");
            return;
        }

        if (send(client_fd, "200 OK", 7 * sizeof(char), 0) <= 0) {
            printf("GET failed!\n");
            return;
        } else {
            printf("GET successful!\n");
            return;
        }
    }
    if (!strcmp(str, "UPLOAD")) {
        if (recv(client_fd, str, 100 * sizeof(char), 0) <= 0) {
            printf("UPLOAD failed!\n");
            return;
        }
```

```
    if (recv(client_fd, str1, 1000 * sizeof(char), 0) <= 0) {
        printf("UPLOAD failed!\n");
        return;
    }
    FILE* fp = fopen(str, "w");
    fputs(str1, fp);
    fclose(fp);
    if (send(client_fd, "200 OK", 7 * sizeof(char), 0) <= 0) {
        printf("UPLOAD failed!\n");
        return;
    } else {
        printf("UPLOAD successful!\n");
        return;
    }
} else if (!strcmp(str, "DOWNLOAD")) {
    if (recv(client_fd, str, 100 * sizeof(char), 0) <= 0) {
        printf("DOWNLOAD failed!\n");
        return;
    }
    FILE* fp = fopen(str, "r");
    if (fp == NULL) {
        str1[0] = '\0';
        if (send(client_fd, str1, sizeof(char), 0) <= 0) {
            printf("DOWNLOAD failed!\n");
            return;
        }
        if (send(client_fd, "400 BAD", 8 * sizeof(char), 0) <= 0) {
            printf("DOWNLOAD failed!\n");
            return;
        }
        return;
    }
    str1[0] = '\0';
    while (fgets(str, 100, fp) != NULL) {
        strcat(str1, str);
    }
    fclose(fp);
    if (send(client_fd, str1, 1000 * sizeof(char), 0) <= 0) {
        printf("DOWNLOAD failed!\n");
        return;
    }
    if (send(client_fd, "200 OK", 7 * sizeof(char), 0) <= 0) {
        printf("DOWNLOAD failed!\n");
        return;
    } else {
        printf("DOWNLOAD successful!\n");
        return;
    }
} else if (!strcmp(str, "RENAME")) {
    if (recv(client_fd, str, 100 * sizeof(char), 0) <= 0) {
        printf("RENAME failed!\n");
```

```
            return;
        }
        if (recv(client_fd, str1, 100 * sizeof(char), 0) <= 0) {
            printf("RENAME failed!\n");
            return;
        }
        if (!rename(str, str1)) {
            if (send(client_fd, "200 OK", 7 * sizeof(char), 0) <= 0) {
                printf("RENAME failed!\n");
                return;
            }
        } else {
            if (send(client_fd, "400 BAD", 8 * sizeof(char), 0) <= 0) {
                printf("RENAME failed!\n");
                return;
            }
        }
        printf("RENAME successful!\n");
        return;
    } else if (!strcmp(str, "DELETE")) {
        if (recv(client_fd, str, 100 * sizeof(char), 0) <= 0) {
            printf("DELETE failed!\n");
            return;
        }
        if (!remove(str)) {
            if (send(client_fd, "200 OK", 7 * sizeof(char), 0) <= 0) {
                printf("DELETE failed!\n");
                return;
            }
        } else {
            if (send(client_fd, "400 BAD", 8 * sizeof(char), 0) <= 0) {
                printf("DELETE failed!\n");
                return;
            }
        }
        printf("DELETE successful!\n");
        return;
    } else {
        if (send(client_fd, "400 BAD", 8 * sizeof(char), 0) <= 0) {
            printf("DELETE failed!\n");
            return;
        }
        printf("Invalid request from client!\n");
        return;
    }
}
void main(int argc, char* argv[]) {
    int PORT;
    if (argc == 2) {
        PORT = atoi(argv[1]);
    } else {
        printf("Enter FTP server port!\n");
```

```
        exit(1);
    }
    int server_fd, client_fd;
    struct sockaddr_in address;
    int addrlen = sizeof(address);
    printf("FTP Server\n");
    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        printf("Socket creation failed!\n");
        exit(1);
    }
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(PORT);
    if (bind(server_fd, (struct sockaddr*) &address, addrlen) < 0) {
        printf("Socket binding failed!\n");
        exit(1);
    }
    if (listen(server_fd, 5) < 0) {
        printf("Listening failed!\n");
        exit(1);
    }
    while (1) {
        if ((client_fd = accept(server_fd, (struct sockaddr*) &address, (socklen_t*) &addrlen)) < 0) {
            printf("Connection failed!\n");
            exit(1);
        } else {
            printf("Connected to client.\n");
        }
        serverRecv(client_fd);
        close(client_fd);
    }
    close(server_fd);
}
```

CLIENT

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <string.h>
void ftp_get(int client_fd) {
    char str[8];
    char str1[1000];
    if (send(client_fd, "GET", 4 * sizeof(char), 0) <= 0) {
        printf("GET failed!\n");
        return;
    }
    if (recv(client_fd, str1, 1000 * sizeof(char), 0) <= 0) {
        printf("GET failed!\n");
        return;
    } else {
        if (recv(client_fd, str, 8 * sizeof(char), 0) <= 0) {
```

```
            printf("GET failed!\n");
            return;
        }
        if (!strcmp(str, "200 OK")) {
            printf("GET successful!\nFiles in FTP server:\n%s", str1);
        } else {
            // Handle failure case
        }
    }
}
void ftp_upload(int client_fd) {
    if (send(client_fd, "UPLOAD", 7 * sizeof(char), 0) <= 0) {
        printf("UPLOAD failed!\n");
        return;
    }
    char filename[100];
    char str[100];
    char str1[1000];
    printf("Enter filename: ");
    scanf("%s", filename);
    FILE* fp = fopen(filename, "r");
    if (fp == NULL) {
        printf("File doesn't exist!\n");
        return;
    }
    if (send(client_fd, filename, 100 * sizeof(char), 0) <= 0) {
        printf("Send failed!\n");
        fclose(fp);
        return;
    }
    str1[0] = '\0';
    while (fgets(str, 100, fp) != NULL) {
        strcat(str1, str);
    }
    if (send(client_fd, str1, 1000 * sizeof(char), 0) <= 0) {
        printf("UPLOAD failed!\n");
        fclose(fp);
        return;
    }
    if (recv(client_fd, str, 7 * sizeof(char), 0) <= 0) {
        printf("UPLOAD failed!\n");
        fclose(fp);
        return;
    }
    if (!strcmp(str, "200 OK")) {
        printf("UPLOAD successful!\n");
    }
    fclose(fp);
}
void ftp_download(int client_fd) {
    if (send(client_fd, "DOWNLOAD", 9 * sizeof(char), 0) <= 0) {
        printf("Send failed!\n");
```

```
        return;
    }
    char filename[100];
    char str[100];
    char str1[1000];
    printf("Enter filename: ");
    scanf("%s", filename);
    if (send(client_fd, filename, 100 * sizeof(char), 0) <= 0) {
        printf("DOWNLOAD failed!\n");
        return;
    }
    if (recv(client_fd, str1, 1000 * sizeof(char), 0) <= 0) {
        printf("DOWNLOAD failed!\n");
        return;
    }
    if (recv(client_fd, str, 100 * sizeof(char), 0) <= 0) {
        printf("DOWNLOAD failed!\n");
        return;
    }
    if (!strcmp(str, "200 OK")) {
        FILE* fp = fopen(filename, "w");
        fputs(str1, fp);
        printf("DOWNLOAD successful!\n");
        fclose(fp);
    } else {
        // Handle failure case
    }
}
void ftp_rename(int client_fd) {
    if (send(client_fd, "RENAME", 7 * sizeof(char), 0) <= 0) {
        printf("RENAME failed!\n");
        return;
    }
    char filename[100];
    char filename1[100];
    char str[8];
    printf("Enter old filename: ");
    scanf("%s", filename);
    printf("Enter new filename: ");
    scanf("%s", filename1);
    if (send(client_fd, filename, 100 * sizeof(char), 0) <= 0) {
        printf("RENAME failed!\n");
        return;
    }
    if (send(client_fd, filename1, 100 * sizeof(char), 0) <= 0) {
        printf("RENAME failed!\n");
        return;
    }
    if (recv(client_fd, str, 8 * sizeof(char), 0) <= 0) {
        printf("RENAME failed!\n");
        return;
    }
```

```
    if (!strcmp(str, "200 OK")) {
        printf("RENAME successful!\n");
    } else {
        // Handle failure case
    }
}
void ftp_delete(int client_fd) {
    if (send(client_fd, "DELETE", 7 * sizeof(char), 0) <= 0) {
        printf("DELETE failed!\n");
        return;
    }
    char filename[100];
    char str[8];
    printf("Enter filename: ");
    scanf("%s", filename);
    if (send(client_fd, filename, 100 * sizeof(char), 0) <= 0) {
        printf("DELETE failed!\n");
        return;
    }
    if (recv(client_fd, str, 8 * sizeof(char), 0) <= 0) {
        printf("DELETE failed!\n");
        return;
    }
    if (!strcmp(str, "200 OK")) {
        printf("DELETE successful!\n");
    } else {
        printf("DELETE failed!\n");
    }
}
void main() {
    int client_fd, PORT;
    struct sockaddr_in serv_addr;
    printf("FTP Client\n");
    printf("Enter FTP server port: ");
    scanf("%d", &PORT);
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = INADDR_ANY;
    serv_addr.sin_port = htons(PORT);
    while (1) {
        client_fd = socket(AF_INET, SOCK_STREAM, 0);
        if (client_fd < 0) {
            printf("Socket creation failed!\n");
            exit(1);
        }
        if (connect(client_fd, (struct sockaddr*) &serv_addr, sizeof(serv_addr)) < 0) {
            printf("Connection failed!\n");
            exit(1);
        }
        int option = 0;
        printf("\n1. GET\n2. UPLOAD\n3. DOWNLOAD\n4. RENAME\n5. DELETE\n6. QUIT\
nEnter your command: ");
```

```
        scanf("%d", &option);
        getchar();
        switch (option) {
            case 1:
                ftp_get(client_fd);
                break;
            case 2:
                ftp_upload(client_fd);
                break;
            case 3:
                ftp_download(client_fd);
                break;
            case 4:
                ftp_rename(client_fd);
                break;
            case 5:
                ftp_delete(client_fd);
                break;
            case 6:
                printf("Exit.\n");
                exit(0);
            default:
                printf("Invalid command!\n");
                break;
        }
        close(client_fd);
    }
}
```

OUTPUT

```
root@DESKTOP-70V9DEP:~# gcc ftps.c -o ftps
root@DESKTOP-70V9DEP:~# ./ftps 8002
FTP Server
Connected to client.
GET successful!
Connected to client.
UPLOAD successful!
Connected to client.
DOWNLOAD successful!
Connected to client.
RENAME successful!
Connected to client.
DELETE successful!
Connected to client.
Connection lost!

root@DESKTOP-70V9DEP:~# gcc ftpc.c
root@DESKTOP-70V9DEP:~# ./a.out
FTP Client
Enter FTP server port: 8002

1. GET
2. UPLOAD
```

3. DOWNLOAD
4. RENAME
5. DELETE
6. QUIT
Enter your command: 1
GET successful!
Files in FTP server:
client25b.c
5c1.c
5cc1.c
client6.c
stopwaits
cons
5cs.c
1. GET
2. UPLOAD
3. DOWNLOAD
4. RENAME
5. DELETE
6. QUIT
Enter your command: 2
Enter filename: d.txt
UPLOAD successful!

1. GET
2. UPLOAD
3. DOWNLOAD
4. RENAME
5. DELETE
6. QUIT
Enter your command: 3
Enter filename: d.txt
DOWNLOAD successful!

1. GET
2. UPLOAD
3. DOWNLOAD
4. RENAME
5. DELETE
6. QUIT
Enter your command: 4
Enter old filename: d.txt
Enter new filename: dd.txt
RENAME successful!

1. GET
2. UPLOAD
3. DOWNLOAD
4. RENAME
5. DELETE
6. QUIT
Enter your command: 5

Enter filename: d.txt
DELETE failed!

1. GET
2. UPLOAD
3. DOWNLOAD
4. RENAME
5. DELETE
6. QUIT
Enter your command: 6
Exit.

IMPLEMENTATION

```
/* NAME : BHAGYA A JAI
   ROLL NO : B21CSB18
   EXPERIMENT : CONCURRENT FILE SERVER*/
```

SERVER

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <string.h>
#include <dirent.h>
void serverRecv(int client_fd) {
   char str[100];
   char str1[1000];
   if(recv(client_fd, str, 100 * sizeof(char), 0) <= 0) {
      printf("Connection lost!\n");
      return;
   }
   FILE* fp = fopen(str, "r");
   int pid = getpid();
   if(fp == NULL) {
      str1[0] = '\0';
      if(send(client_fd, str1, sizeof(char), 0) <= 0) {
         printf("GET failed!\n");
         return;
      }
      if(send(client_fd, &pid, sizeof(int), 0) <= 0) {
         printf("GET failed!\n");
         return;
      }
      if(send(client_fd, "400 BAD", 8 * sizeof(char), 0) <= 0) {
         printf("GET failed!\n");
         return;
      }
      return;
   }
   str1[0] = '\0';
   while(fgets(str, 100, fp) != NULL) {
      strcat(str1, str);
   }
   fclose(fp);
   if(send(client_fd, str1, 1000 * sizeof(char), 0) <= 0) {
      printf("GET failed!\n");
      return;
   }
   if(send(client_fd, &pid, sizeof(int), 0) <= 0) {
      printf("GET failed!\n");
      return;
   }
   if(send(client_fd, "200 OK", 7 * sizeof(char), 0) <= 0) {
      printf("GET failed!\n");
```

```
        return;
    } else {
        printf("GET successful!\n");
        return;
    }
}
void main(int argc, char* argv[]) {
    int PORT;
    if(argc == 2) {
        PORT = atoi(argv[1]);
    } else {
        printf("Enter file server port!\n");
        exit(1);
    }
    int server_fd, client_fd;
    struct sockaddr_in address;
    int addrlen = sizeof(address);
    printf("Concurrent File Server\n");
    if((server_fd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        printf("Socket creation failed!\n");
        exit(1);
    }
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(PORT);
    if(bind(server_fd, (struct sockaddr*) &address, addrlen) < 0) {
        printf("Socket binding failed!\n");
        exit(1);
    }
    if(listen(server_fd, 5) < 0) {
        printf("Listening failed!\n");
        exit(1);
    }
    while(1) {
        if((client_fd = accept(server_fd, (struct sockaddr*) &address, (socklen_t*) &addrlen)) < 0) {
            printf("Connection failed!\n");
            exit(1);
        } else {
            printf("Connected to client.\n");
        }
        if(fork() == 0) {
            serverRecv(client_fd);
            close(client_fd);
        }
    }
    close(server_fd);
}
```

CLIENT

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <arpa/inet.h>
```

```
#include <unistd.h>
#include <string.h>
void get(int client_fd) {
    char filename[100];
    char str[100];
    char str1[1000];
    printf("Enter filename: ");
    scanf("%s", filename);
    if (send(client_fd, filename, 100 * sizeof(char), 0) <= 0) {
        printf("GET failed!\n");
        return;
    } else {
        if (recv(client_fd, str1, 1000 * sizeof(char), 0) <= 0) {
            printf("GET failed!\n");
            return;
        }
        int pid;
        if (recv(client_fd, &pid, sizeof(int), 0) <= 0) {
            printf("GET failed!\n");
            return;
        }
        if (recv(client_fd, str, 100 * sizeof(char), 0) <= 0) {
            printf("GET failed!\n");
            return;
        }
        if (!strcmp(str, "200 OK")) {
            FILE* fp = fopen(filename, "w");
            fputs(str1, fp);
            printf("GET successful (PID %d)!\n", pid);
            fclose(fp);
        } else {
            printf("GET failed (PID %d)!\n", pid);
        }
    }
}
void main() {
    int client_fd, PORT;
    struct sockaddr_in serv_addr;
    printf("File Client\n");
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = INADDR_ANY;
    while (1) {
        printf("Enter file server port: ");
        scanf("%d", &PORT);
        serv_addr.sin_port = htons(PORT);
        client_fd = socket(AF_INET, SOCK_STREAM, 0);
        if (client_fd < 0) {
            printf("Socket creation failed!\n");
            exit(1);
        }
        if (connect(client_fd, (struct sockaddr*) &serv_addr, sizeof(serv_addr)) < 0) {
            printf("Connection failed!\n");
```

```
            exit(1);
        }
        get(client_fd);
        close(client_fd);
    }
}
```

OUTPUT

```
root@DESKTOP-70V9DEP:~# gcc cons.c -o cons
root@DESKTOP-70V9DEP:~# ./cons 8000
Concurrent File Server
Connected to client.
GET successful!
Connected to client.

root@DESKTOP-70V9DEP:~# gcc conc.c
root@DESKTOP-70V9DEP:~# ./a.out
File Client
Enter file server port: 8000
Enter filename: d.txt
GET successful (PID 7287)!
Enter file server port: 8000
Enter filename: abc.de
GET failed (PID 7288)!
Enter file server port: "C
```

IMPLEMENTATION

```
/* NAME : BHAGYA A JAI
   ROLL NO : B21CSB18
   EXPERIMENT : LEAKY  BUCKET ALORITHM*/
```

SERVER

```
 #include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
void main() {
    int server_fd, client_fd;
    struct sockaddr_in address;
    int PORT, addrlen = sizeof(address);
    printf("Enter port: ");
    scanf("%d", &PORT);
    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        printf("Socket creation failed!\n");
        exit(1);
    }
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(PORT);
    if (bind(server_fd, (struct sockaddr*) &address, addrlen) < 0) {
        printf("Socket binding failed!\n");
        exit(1);
    }
    if (listen(server_fd, 1) < 0) {
        printf("Listening failed!\n");
        exit(1);
    }
    if ((client_fd = accept(server_fd, (struct sockaddr*) &address, (socklen_t*) &addrlen)) < 0) {
        printf("Connection failed!\n");
        exit(1);
    }
    int n;
    while (1) {
        int k = recv(client_fd, &n, sizeof(int), 0);
        if (k < 0) {
            printf("Receive failed!\n");
            break;
        } else if (k == 0) {
            printf("Receive complete!\n");
            break;
        } else {
            printf("Received: %d\n", n);
        }
    }
    close(server_fd);
    close(client_fd);
}
```

CLIENT

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
typedef struct queue {
    int* arr;
    int FRONT;
    int REAR;
    int SIZE;
    int REMAINING;
} queue;
void initQueue(queue* q, int size) {
    q->FRONT = -1;
    q->REAR = -1;
    q->SIZE = size;
    q->REMAINING = size;
    q->arr = malloc(size * sizeof(int));
}
void enqueue(queue* q, int i) {
    if (q->FRONT == -1)
        q->FRONT = 0;
    else if ((q->REAR + 1) % q->SIZE == q->FRONT)
        return;

    q->REAR = (q->REAR + 1) % q->SIZE;
    q->arr[q->REAR] = i;
}
int dequeue(queue* q) {
    int data;
    if (q->FRONT == -1)
        return -1;
    else if (q->FRONT == q->REAR) {
        data = q->arr[q->FRONT];
        q->FRONT = -1;
        q->REAR = -1;
    } else {
        data = q->arr[q->FRONT];
        q->FRONT = (q->FRONT + 1) % q->SIZE;
    }
    q->REMAINING++;
    return data;
}
void sendData(int client_fd, queue* q, int size) {
    int k = 0, n;
    while (1) {
        printf("\n");
        for (int i = 0; i < size; i++) {
            n = dequeue(q);
            if (n == -1)
                return;
            else if (n == 0)
```

```
            k++;
            if (send(client_fd, &n, sizeof(int), 0) < 0) {
                printf("Send failed!\n");
                exit(1);
            }
        }
        sleep(1);
    }
    printf("Sent packet %d: %d\n", k, n);
}
void main(int argc, char* argv[]) {
    int PORT, client_fd;
    struct sockaddr_in serv_addr;
    printf("Enter port: ");
    scanf("%d", &PORT);
    client_fd = socket(AF_INET, SOCK_STREAM, 0);
    if (client_fd < 0) {
        printf("Socket creation failed!\n");
        exit(1);
    }
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = INADDR_ANY;
    serv_addr.sin_port = htons(PORT);
    if (connect(client_fd, (struct sockaddr*) &serv_addr, sizeof(serv_addr)) < 0) {
        printf("Connection failed!\n");
        exit(1);
    }
    queue q;
    int num, size;
    printf("Enter bucket size: ");
    scanf("%d", &size);
    initQueue(&q, size);
    printf("Enter packet size to send per second: ");
    scanf("%d", &size);
    while (1) {
        printf("\nEnter number of packets to send: ");
        scanf("%d", &num);
        int packets[num];
        for (int i = 0; i < num; i++) {
            printf("Enter packet %d size: ", i + 1);
            scanf("%d", &packets[i]);
        }
        for (int i = 0; i < num; i++) {
            if (q.REMAINING < packets[i]) {
                printf("\nBucket full! Packet %d rejected!\n", i + 1);
                continue;
            }
            for (int j = 0; j < packets[i]; j++) {
                enqueue(&q, j);
                q.REMAINING--;
            }
        }
```

```
        sendData(client_fd, &q, size);
    }
    close(client_fd);
}
```

OUTPUT

```
root@DESKTOP-70V9DEP:~# gcc leakys.c
root@DESKTOP-70V9DEP:~# ./a.out
Server
Enter port: 8000
Received: 0
Received: 1
Received: 2
Received: 3
Received: 0
Received: 1
Received: 2
Received: 3

Received: 4
Received: 5
Received: 6
Received: 7
Received: 0
Received: 1
Received: 0
Received: 1
Received: 2
Received: 3
Received: 4
Received: 0
Received: 1
Received: 2
Received: 0
Received: 1
Receive complete!

root@DESKTOP-70V9DEP:~# gcc leakyc.c
root@DESKTOP-70V9DEP:~# ./a.out
Client
Enter port: 8000
Enter bucket size: 20
Enter packet size to send per second: 3
Enter number of packets to send: 5
Enter packet 1 size: 4
Enter packet 2 size: 8
Enter packet 3 size: 2
Enter packet 4 size: 5
Enter packet 5 size: 3
Bucket full: Packet 5 rejected!
Sent packet 1:0
Sent packet 1: 1
```

Sent packet 1: 2
Sent packet 1: 3
Sent packet 2: 0
Sent packet 2: 1
Sent packet 2: 2
Sent packet 2: 3
Sent packet 2: 4
Sent packet 2: 5
Sent packet 2: 6
Sent packet 2: 7
Sent packet 3: 0
Sent packet 3: 1
Sent packet 4: 0
Sent packet 4: 1
Sent packet 4: 2
Sent packet 4: 3
Sent packet 4: 4
Enter number of packets to send: 2
Enter packet 1 size: 3
Enter packet 2 size: 2
Sent packet 1: 0
Sent packet 1: 1
Sent packet 1: 2
Sent packet 2: 0
Sent packet 2: 1
Enter number of packets to send: ^C