STUDENT PORTFOLIO

Name : Vishal Aravinth A
Register number : RA2011031010013
Mail ID : va0609@srmist.edu.in
Department : CSE
Specialization : Information Technology
Semester : 5

Subject Title: 18CSC302J Computer Networks

Lab Excercises

Experiment no : 3 **Simple TCP/IP Client Server Communication**

Server code:

```
#include <stdio.h>
#include <netdb.h>
#include <netinet/in.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#define MAX 80
#define PORT 8083
#define SA struct sockaddr

void func(int connfd)
{
	char buff[MAX];
	int n;

	for (;;) {
		bzero(buff, MAX);

		read(connfd, buff, sizeof(buff));

		printf("From client: %s\t To client : ", buff);
		bzero(buff, MAX);
		n = 0;

		while ((buff[n++] = getchar()) != '\n')
			;
```

```
                write(connfd, buff, sizeof(buff));

                if (strncmp("exit", buff, 4) == 0) {
                        printf("Server Exit...\n");
                        break;
                }
        }
}

int main()
{
        int sockfd, connfd, len;
        struct sockaddr_in servaddr, cli;

        sockfd = socket(AF_INET, SOCK_STREAM, 0);
        if (sockfd == -1) {
                printf("socket creation failed...\n");
                exit(0);
        }
        else
                printf("Socket successfully created..\n");
        bzero(&servaddr, sizeof(servaddr));

        servaddr.sin_family = AF_INET;
        servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
        servaddr.sin_port = htons(PORT);

        if ((bind(sockfd, (SA*)&servaddr, sizeof(servaddr))) != 0) {
                printf("socket bind failed...\n");
```

```
        exit(0);
    }
    else
        printf("Socket successfully binded..\n");

    if ((listen(sockfd, 5)) != 0) {
        printf("Listen failed...\n");
        exit(0);
    }
    else
        printf("Server listening..\n");
    len = sizeof(cli);

    connfd = accept(sockfd, (SA*)&cli, &len);
    if (connfd < 0) {
        printf("server accept failed...\n");
        exit(0);
    }
    else
        printf("server accept the client...\n");

    func(connfd);

    close(sockfd);
}
```

Experiment no : 3 **Simple TCP/IP Client Server Communication**

Client code:

```
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#define MAX 80
#define PORT 8083
#define SA struct sockaddr
void func(int sockfd)
{
	char buff[MAX];
	int n;
	for (;;) {
		bzero(buff, sizeof(buff));
		printf("Enter the string : ");
		n = 0;
		while ((buff[n++] = getchar()) != '\n')
			;
		write(sockfd, buff, sizeof(buff));
		bzero(buff, sizeof(buff));
		read(sockfd, buff, sizeof(buff));
		printf("From Server : %s", buff);
		if ((strncmp(buff, "exit", 4)) == 0) {
			printf("Client Exit...\n");
			break;
		}
	}
}
```

Experiment no : 3 Simple TCP/IP Client Server Communication

```
int main()
{
	int sockfd, connfd;
	struct sockaddr_in servaddr, cli;

	sockfd = socket(AF_INET, SOCK_STREAM, 0);
	if (sockfd == -1) {
		printf("socket creation failed...\n");
		exit(0);
	}
	else
		printf("Socket successfully created..\n");
	bzero(&servaddr, sizeof(servaddr));

	servaddr.sin_family = AF_INET;
	servaddr.sin_addr.s_addr = inet_addr("127.0.0.1");
	servaddr.sin_port = htons(PORT);

	if (connect(sockfd, (SA*)&servaddr, sizeof(servaddr)) != 0) {
		printf("connection with the server failed...\n");
		exit(0);
	}
	else
		printf("connected to the server..\n");

	func(sockfd);

	close(sockfd);
}
```

Experiment no : 3 **Simple TCP/IP Client Server Communication**

Output:

Client side output:

Socket successfully created..

connected to the server..

Enter the string : Hi

From Server : hii

Enter the string : how are you!?

From Server : im good

Enter the string :

Server side output:

Socket successfully created..

Socket successfully binded..

Server listening..

server accept the client...

From client: Hi

To client : hii

From client: how are you!?

To client : im good

Experiment No: 4 UDP Echo Client Server Communication

Server code:

```
// Server side implementation of UDP client-server model
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>

#define PORT   8080
#define MAXLINE 1024
int main() {
        int sockfd;
        char buffer[MAXLINE];
        char *hello = "Hello from server";
        struct sockaddr_in servaddr, cliaddr;
        if ( (sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0 ) {
                perror("socket creation failed");
                exit(EXIT_FAILURE);
        }

        memset(&servaddr, 0, sizeof(servaddr));
        memset(&cliaddr, 0, sizeof(cliaddr));
        servaddr.sin_family = AF_INET; // IPv4
        servaddr.sin_addr.s_addr = INADDR_ANY;
        servaddr.sin_port = htons(PORT);
        if ( bind(sockfd, (const struct sockaddr *)&servaddr,
```

```
                        sizeof(servaddr)) < 0 )
        {
                perror("bind failed");
                exit(EXIT_FAILURE);
        }

        int len, n;

        len = sizeof(cliaddr);

        n = recvfrom(sockfd, (char *)buffer, MAXLINE,
                                MSG_WAITALL, ( struct sockaddr *) &cliaddr,
                                &len);
        buffer[n] = '\0';
        printf("Client : %s\n", buffer);
        sendto(sockfd, (const char *)hello, strlen(hello),
                MSG_CONFIRM, (const struct sockaddr *) &cliaddr,
                        len);
        printf("Hello message  sent.\n");

        return 0;
}
```

Client code:

```
// Client side implementation of UDP client-server model
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
```

Experiment No: 4 UDP Echo Client Server Communication

```
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>

#define PORT   8080
#define MAXLINE 1024

// Driver code
int main() {
        int sockfd;
        char buffer[MAXLINE];
        char *hello = "Hello from client";
        struct sockaddr_in      servaddr;

        // Creating socket file descriptor
        if ( (sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0 ) {
                perror("socket creation failed");
                exit(EXIT_FAILURE);
        }

        memset(&servaddr, 0, sizeof(servaddr));

        // Filling server information
        servaddr.sin_family = AF_INET;
        servaddr.sin_port = htons(PORT);
        servaddr.sin_addr.s_addr = INADDR_ANY;

        int n, len;
```

```
        sendto(sockfd, (const char *)hello, strlen(hello),
                MSG_CONFIRM, (const struct sockaddr *) &servaddr,
                        sizeof(servaddr));
        printf("Hello message sent.\n");

        n = recvfrom(sockfd, (char *)buffer, MAXLINE,
                                MSG_WAITALL, (struct sockaddr *) &servaddr,
                                &len);
        buffer[n] = '\0';
        printf("Server : %s\n", buffer);

        close(sockfd);
        return 0;
}
```

Output:

Client side output:

Running /home/ubuntu/environment/RA2011031010013/exp-4_client.c

Hello message sent.

Server : Hello from server

Server side output:

Running /home/ubuntu/environment/RA2011031010013/exp-4_server.c

Client : Hello from client

Hello message sent.

Experiment No: 5 **Concurrent TCP/IP Day-Time Server**

Server code:

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

int main()
{
   int sfd,r,bi,port;
   char buff[1024];
   struct sockaddr_in servaddr,cliaddr;
   socklen_t clilen;
   sfd=socket(AF_INET,SOCK_DGRAM,0);
   if(sfd==-1)
   {
      perror("Socket");
      return 0;
   }

   servaddr.sin_family = AF_INET;
   servaddr.sin_port = htons(12345);
   servaddr.sin_addr.s_addr = INADDR_ANY;

   bi=bind(sfd,(struct sockaddr*)&servaddr,sizeof(servaddr));
   if(bi==-1)
   {
      perror("Bind()");
      return 0;
```

```
    }
    printf("server is running.....\n");
    clilen = sizeof(cliaddr);
    r=recvfrom(sfd,buff,sizeof(buff),0,(struct sockaddr*)&cliaddr,&clilen);
    buff[r]=0;
    time_t ticks;
    ticks = time(NULL);
    snprintf(buff,sizeof(buff),"%24s\r\n",ctime(&ticks));
    sendto(sfd,buff,sizeof(buff),0,(struct sockaddr*)&cliaddr,sizeof(cliaddr));
    exit(0);
    return 0;
}
```

Client code:

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

int main()
{
    int listenfd,port,r;
    char buff[1024];
    struct sockaddr_in servaddr,cliaddr;
    socklen_t servlen;
    listenfd = socket(AF_INET,SOCK_DGRAM,0);
    if(listenfd==-1)
    {
        perror("Socket");
```

```
    return 0;
  }

  servaddr.sin_family = AF_INET;
  servaddr.sin_port = htons(12345);
  servaddr.sin_addr.s_addr = INADDR_ANY;
  sendto(listenfd,buff,sizeof(buff),0,(struct sockaddr*)&servaddr,sizeof(servaddr));
  r=recvfrom(listenfd,buff,sizeof(buff),0,(struct sockaddr*)&servaddr,&servlen);
  buff[r]=0;
  printf("\n The time received from the server:%s\n",buff);
  exit(0);
  return 0;
}
```

Output:

Client side output:

Running /home/ubuntu/environment/RA2011031010013/exp-5_server.c

server is running.....

Server side output:

Running /home/ubuntu/environment/RA2011031010013/exp-5_client.c

The time received from the server:Wed Nov 9 08:35:35 2022

Experiment No: 6 **Half Duplex Chat Using TCP/IP**

Server code:

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

void func(int cfd) {
   char buff1[1024];
   char buff2[1024];
   int n;

   for( ; ; ) {
      bzero(buff1,sizeof(buff1),0);
      printf("Enter the message: ");
      gets(buff1);
      send(cfd,buff1,sizeof(buff1),0);

      if(strncmp("exit",buff2,4) == 0){
         printf("Server exiting....\n");
         break;
      }

      bzero(buff2,sizeof(buff2));
      read(cfd,buff2,sizeof(buff2),0);
      printf("Client to server: %s\n",buff2);
```

```
    }
}

int main() {
    int sfd,bi,cfd;
    struct sockaddr_in servaddr,cliaddr;
    socklen_t clilen;

    sfd = socket(AF_INET,SOCK_STREAM,0);
    if(sfd == -1){
        perror("Socket cannot be created....\n");
        exit(0);
    }
    printf("Server is running....\n");
    bzero(&servaddr,sizeof(servaddr));

    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons(9993);
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);

    bi = bind(sfd,(struct sockaddr*)&servaddr,sizeof(servaddr));
    if(bi == -1){
        perror("Bind function failed.....\n");
        exit(0);
    }

    printf("Socket binded....\n");
    if((listen(sfd, 5) != 0)){
        printf("Listen action failed...\n");
```

```
        exit(0);
    }
    printf("Server is listening....\n");
    clilen = sizeof(cliaddr);
    cfd = accept(sfd,NULL,NULL);
    if(cfd < 0) {
        printf("Server accept failed...\n");
        exit(0);
    }

    func(cfd);
    close(sfd);
}
```

Client code:

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

#define h_addr h_addr_list[0]

void func(int sockfd)
{
```

```
        char buff1[1024];
        char buff2[1024];
        int n;
        for (;;) {
                bzero(buff1, sizeof(buff1));
                recv(sockfd, buff1, sizeof(buff1),0);
                printf("From Server : %s\n", buff1);
                if ((strncmp(buff1, "exit", 4)) == 0) {
                        printf("Client Exiting...\n");
                        break;
                }

                bzero(buff2, sizeof(buff2));
                printf("Enter the message : ");
                gets(buff2);
                send(sockfd, buff2, sizeof(buff2),0);
        }
}


int main() {
   int sfd, cfd;
        struct sockaddr_in servaddr, cliaddr;

        sfd = socket(AF_INET, SOCK_STREAM, 0);
        if (sfd == -1) {
                printf("socket creation failed...\n");
                exit(0);
        }
```

```
        else
                printf("Socket successfully created..\n");
        char hostname[1024], ipaddress[1024];
   struct hostent *hostIP;

   if(gethostname(hostname,sizeof(hostname))==0){
      hostIP = gethostbyname(hostname);
   }else{
      printf("IP Address Not ");
   }

        servaddr.sin_family = AF_INET;
        servaddr.sin_addr.s_addr = INADDR_ANY;
        servaddr.sin_port = htons(9993);

        if (connect(sfd, (struct sockaddr*)&servaddr, sizeof(servaddr)) != 0) {
                printf("connection with the server failed...\n");
                exit(0);
        }
        else
                printf("connected to the server..\n");


        func(sfd);

        close(sfd);
}
```

Experiment No: 6 **Half Duplex Chat Using TCP/IP**

Output:

Client side output:

Socket successfully created..

connected to the server..

From Server : Hey this is the server

Enter the message : Hi this is the client

From Server : wut communication mode is this?

Enter the message : we are on half duplex communication

Server side outpt:

Server is running....

Socket binded....

Server is listening....

Enter the message: Hey this is the server

Client to server: Hi this is the client

Enter the message: wut communication mode is this?

Client to server: we are on half duplex communication

Enter the message:

Experiment No:7 **FULL-DUPLEX CHAT USING TCP-IP**

Server Code:

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

int main(int argc,char *argv[])
{
int clientSocketDescriptor,socketDescriptor;

struct sockaddr_in serverAddress,clientAddress;
socklen_t clientLength;

char recvBuffer[1000],sendBuffer[1000];
pid_t cpid;
bzero(&serverAddress,sizeof(serverAddress));
serverAddress.sin_family=AF_INET;
serverAddress.sin_addr.s_addr=htonl(INADDR_ANY);
serverAddress.sin_port=htons(5949);
socketDescriptor=socket(AF_INET,SOCK_STREAM,0);
```

Experiment No:7 FULL-DUPLEX CHAT USING TCP-IP

```
bind(socketDescriptor,(struct sockaddr*)&serverAddress,sizeof(serverAddress));

listen(socketDescriptor,5);

printf("%s\n","Server is running ...");

clientSocketDescriptor=accept(socketDescriptor,(struct
sockaddr*)&clientAddress,&clientLength);

cpid=fork();


if(cpid==0)

{

while(1)

{

bzero(&recvBuffer,sizeof(recvBuffer));

recv(clientSocketDescriptor,recvBuffer,sizeof(recvBuffer),0);

printf("\nCLIENT : %s\n",recvBuffer);

}

}

else

{

while(1)

{


bzero(&sendBuffer,sizeof(sendBuffer));

printf("\nType a message here ...  ");

fgets(sendBuffer,10000,stdin);
```

```
send(clientSocketDescriptor,sendBuffer,strlen(sendBuffer)+1,0);
printf("\nMessage sent !\n");
}
}
return 0;
}
```

Client Code:

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

int main()
{
int socketDescriptor;

struct sockaddr_in serverAddress;
char sendBuffer[1000],recvBuffer[1000];
```

Experiment No:7 FULL-DUPLEX CHAT USING TCP-IP

```
pid_t cpid;

bzero(&serverAddress,sizeof(serverAddress));
serverAddress.sin_family=AF_INET;
serverAddress.sin_addr.s_addr=inet_addr("127.0.0.1");
serverAddress.sin_port=htons(5949);
socketDescriptor=socket(AF_INET,SOCK_STREAM,0);
connect(socketDescriptor,(struct sockaddr*)&serverAddress,sizeof(serverAddress));
cpid=fork();
if(cpid==0)
{
while(1)
{
bzero(&sendBuffer,sizeof(sendBuffer));
printf("\nType a message here  ");
fgets(sendBuffer,10000,stdin);
send(socketDescriptor,sendBuffer,strlen(sendBuffer)+1,0);
printf("\nMessage sent !\n");
}
}
else
{
while(1)
```

```
{
bzero(&recvBuffer,sizeof(recvBuffer));
recv(socketDescriptor,recvBuffer,sizeof(recvBuffer),0);
printf("\nSERVER : %s\n",recvBuffer);
}
}
return 0;
}
```

Output:

Client Side:

Type a message here HELLO i AM CLIENT

Message sent !

Type a message here

SERVER : HELLO

Server Side:

Server is running ...

Type a message here ...

CLIENT : HELLO i AM CLIENT

HELLO

Message sent !

Type a message here ...

Experiment No: 8 **Implementation of File Transfer Protocol**

Server code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <arpa/inet.h>
#define SIZE 1024

void write_file(int sockfd){
  int n;
  FILE *fp;
  char *filename = "receivedv.txt";
  char buffer[SIZE];
  fp = fopen(filename, "w");
  while (1) {
    n = recv(sockfd, buffer, SIZE, 0);
    if (n <= 0){
      break;
      return;
    }
    fprintf(fp, "%s", buffer);
    bzero(buffer, SIZE);
  }
  return;
}

int main(){
  char *ip = "127.0.0.1";
  int port = 6666;
  int e;
```

```
 int sockfd, new_sock;
 struct sockaddr_in server_addr, new_addr;
 socklen_t addr_size;
 char buffer[SIZE];

 sockfd = socket(AF_INET, SOCK_STREAM, 0);
 if(sockfd < 0) {
  perror("[-]Error in socket");
  exit(1);
 }
 printf("[+]Server socket created successfully.\n");
 server_addr.sin_family = AF_INET;
 server_addr.sin_port = port;
 server_addr.sin_addr.s_addr = inet_addr(ip);
 e = bind(sockfd, (struct sockaddr*)&server_addr, sizeof(server_addr));
 if(e < 0) {
  perror("[-]Error in bind");
  exit(1);
 }
 printf("[+]Binding successfull.\n");
 if(listen(sockfd, 10) == 0){
  printf("[+]Listening....\n");
}else{
  perror("[-]Error in listening");
  exit(1);
}

 addr_size = sizeof(new_addr);
 new_sock = accept(sockfd, (struct sockaddr*)&new_addr, &addr_size);
```

```
  write_file(new_sock);
  printf("[+]Data written in the file successfully.\n");
  return 0;
}
```

Client code:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <arpa/inet.h>
#define SIZE 1024

void send_file(FILE *fp, int sockfd){
  int n;
  char data[SIZE] = {0};
  while(fgets(data, SIZE, fp) != NULL) {
    if (send(sockfd, data, sizeof(data), 0) == -1) {
      perror("[-]Error in sending file.");
      exit(1);
    }
    bzero(data, SIZE);
  }
}

int main(){
  char *ip = "127.0.0.1";
  int port = 6666;
  int e;
```

```
 int sockfd;
 struct sockaddr_in server_addr;
 FILE *fp;
 char *filename = "RA2011031010013.txt";

 sockfd = socket(AF_INET, SOCK_STREAM, 0);
 if(sockfd < 0) {
  perror("[-]Error in socket");
  exit(1);
 }
 printf("[+]Server socket created successfully.\n");
 server_addr.sin_family = AF_INET;
 server_addr.sin_port = port;
 server_addr.sin_addr.s_addr = inet_addr(ip);
 e = connect(sockfd, (struct sockaddr*)&server_addr, sizeof(server_addr));
if(e == -1) {
  perror("[-]Error in socket");
  exit(1);
 }
printf("[+]Connected to Server.\n");
 fp = fopen(filename, "r");
 if (fp == NULL) {
  perror("[-]Error in reading file.");
  exit(1);
 }
 send_file(fp, sockfd);
 printf("[+]File data sent successfully.\n");
 printf("[+]Closing the connection.\n");
 close(sockfd);
```

```
  return 0;
}
```

Output:

Client side output:

[+]Server socket created successfully.

[+]Connected to Server.

[+]File data sent successfully.

[+]Closing the connection.

Server side output:

[+]Server socket created successfully.

[+]Binding successfull.

[+]Listening....

[+]Data written in the file successfully.

Client side file:

Hi, how are you!??

Experiment no:9 **REMOTE COMMAND EXECUTION USING UDP**

Server code:

```
#include <sys/types.h>
#include <sys/socket.h>
#include <stdio.h>
#include <stdlib.h>
#include <netdb.h>
#include <netinet/in.h>
#include <string.h>
#include <sys/stat.h>
#include <arpa/inet.h>
#include <unistd.h>
#define MAX 1000
int main()
{
   int serverDescriptor = socket(AF_INET, SOCK_DGRAM, 0);
   int size;
   char buffer[MAX], message[] = "Command Successfully executed !";
   struct sockaddr_in clientAddress, serverAddress;
   socklen_t clientLength = sizeof(clientAddress);
   bzero(&serverAddress, sizeof(serverAddress));
   serverAddress.sin_family = AF_INET;
   serverAddress.sin_addr.s_addr = htonl(INADDR_ANY);
   serverAddress.sin_port = htons(8099);
   bind(serverDescriptor, (struct sockaddr *)&serverAddress, sizeof(serverAddress));
```

```
   while (1)
   {
      bzero(buffer, sizeof(buffer));
      recvfrom(serverDescriptor, buffer, sizeof(buffer), 0, (struct sockaddr
*)&clientAddress,&clientLength);
      system(buffer);
      printf("Command Executed ... %s ", buffer);
      sendto(serverDescriptor, message, sizeof(message), 0, (struct sockaddr
*)&clientAddress,clientLength);
   }
   close(serverDescriptor);
   return 0;
}
```

Client code:

```
#include <sys/types.h>
#include <sys/socket.h>
#include <stdio.h>
#include <stdlib.h>
#include <netdb.h>
#include <netinet/in.h>
#include <string.h>
#include <sys/stat.h>
#include <arpa/inet.h>
#include <unistd.h>
#define MAX 1000
```

Experiment no:9 REMOTE COMMAND EXECUTION USING UDP

```
int main()
{
   int serverDescriptor = socket(AF_INET, SOCK_DGRAM, 0);
   char buffer[MAX], message[MAX];
   struct sockaddr_in cliaddr, serverAddress;
   socklen_t serverLength = sizeof(serverAddress);
   bzero(&serverAddress, sizeof(serverAddress));
   serverAddress.sin_family = AF_INET;
   serverAddress.sin_addr.s_addr = inet_addr("127.0.0.2");
   serverAddress.sin_port = htons(8099);
   bind(serverDescriptor, (struct sockaddr *)&serverAddress, sizeof(serverAddress));
   while (1)
   {
      printf("\nCOMMAND FOR EXECUTION ... ");
      fgets(buffer, sizeof(buffer), stdin);
      sendto(serverDescriptor, buffer, sizeof(buffer), 0, (struct sockaddr *)&serverAddress,
      serverLength);
      printf("\nData Sent!");
      recvfrom(serverDescriptor, message, sizeof(message), 0, (struct
sockaddr*)&serverAddress, &serverLength);
      printf("UDP SERVER : %s", message);
   }
   return 0;
}
```

Experiment no:9 REMOTE COMMAND EXECUTION USING UDP

Output:

Client side output:

COMMAND FOR EXECUTION ... date

Data Sent!UDP SERVER : Command Successfully executed !

COMMAND FOR EXECUTION ... ls

Data Sent!UDP SERVER : Command Successfully executed !

ata Sent!UDP SERVER : Command Successfully executed !

COMMAND FOR EXECUTION ... cal

Data Sent!UDP SERVER : Command Successfully executed !

Server side output:

Wed Nov 9 08:25:08 UTC 2022

Command Executed ... date

RA2011031010015.txt 'exp-10 server.c.o' exp-4server.c exp-5server.c.o 'exp-7 client.c' 'exp-8 client.c.o' exp-9sever.c

Command Executed ... ls

November 2022

Su Mo Tu We Th Fr Sa

1 2 3 4 5

6 7 8 9 10 11 12

13 14 15 16 17 18 19

20 21 22 23 24 25 26

27 28 29 30

Command Executed ... cal

Experiment No:10 **ARP IMPLEMENTATIION**

Code:

```
#include<sys/types.h>
#include<sys/socket.h>
#include<net/if_arp.h>
#include<sys/ioctl.h>
#include<stdio.h>
#include<string.h>
#include<unistd.h>
#include<math.h>
#include<complex.h>
#include<arpa/inet.h>
#include<netinet/in.h>
#include<netinet/if_ether.h>
#include<net/ethernet.h>
#include<stdlib.h>
int main()
{
    struct sockaddr_in sin={0};
    struct arpreq myarp={{0}};
    unsigned char *ptr;
    int sd;
    sin.sin_family=AF_INET;
    printf("Enter IP address: ");
    char ip[20];
```

```
scanf("%s", ip);
if(inet_pton(AF_INET,ip,&sin.sin_addr)==0)
{
   printf("IP address Entered '%s' is not valid \n",ip);
   exit(0);
}
memcpy(&myarp.arp_pa,&sin,sizeof(myarp.arp_pa));
strcpy(myarp.arp_dev,"echo");
sd=socket(AF_INET,SOCK_DGRAM,0);
printf("\nSend ARP request\n");
if(ioctl(sd,SIOCGARP,&myarp)==1)
{
   printf("No Entry in ARP cache for '%s'\n",ip);
   exit(0);
}
ptr=&myarp.arp_pa.sa_data[0];
printf("Received ARP Reply\n");
printf("\nMAC Address for '%s' : ",ip);
printf("%p:%p:%p:%p:%p:%p\n",ptr,(ptr+1),(ptr+2),(ptr+3),(ptr+4),(ptr+5));
return 0;
}
```

Experiment No:10 **ARP IMPLEMENTATIION**

Output:

Enter IP address: 128.16.2.0

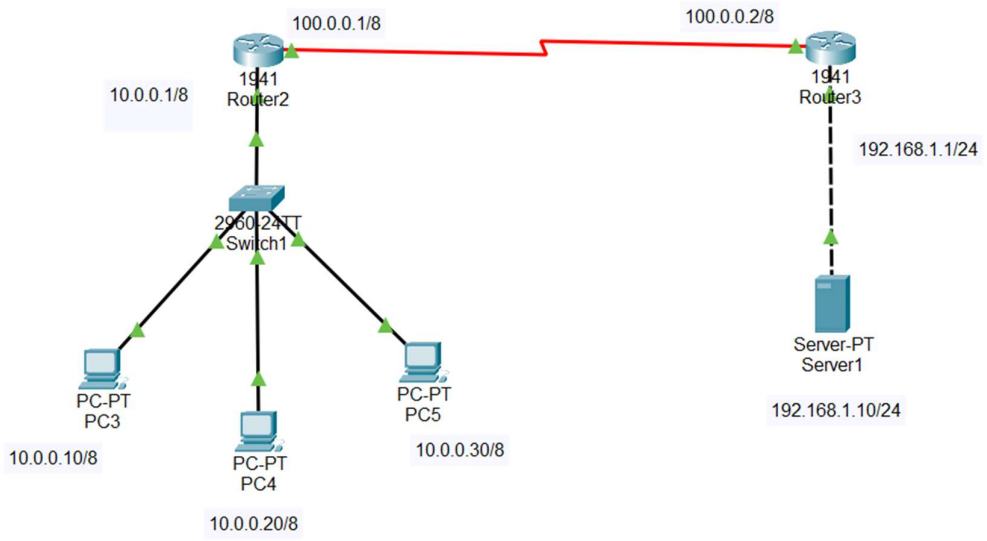
Send ARP request

Received ARP Reply

MAC Address for '128.16.2.0' : 0x7ffe9c24c492:0x7ffe9c24c493:0x7ffe9c24c494:0x7ffe9c24c495:0x7ffe9c24c496:0x7ffe9c24c497

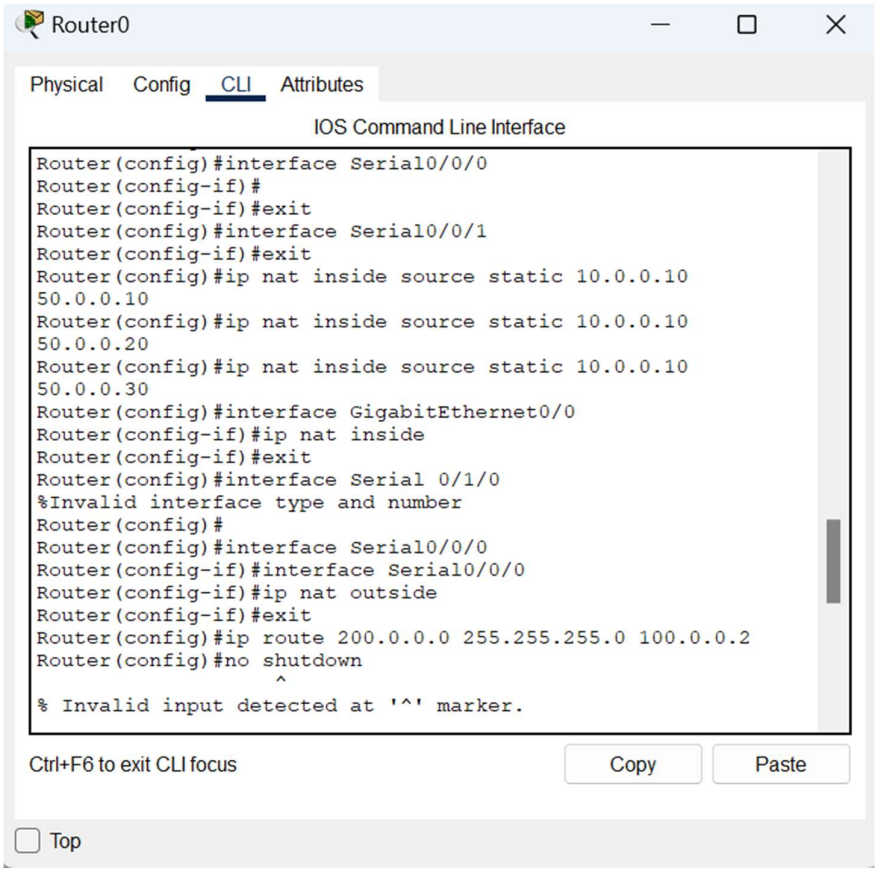
EXPERIMENT NO:12 **Implementation of Network Address Translation**

Cisco Packet Tracer:



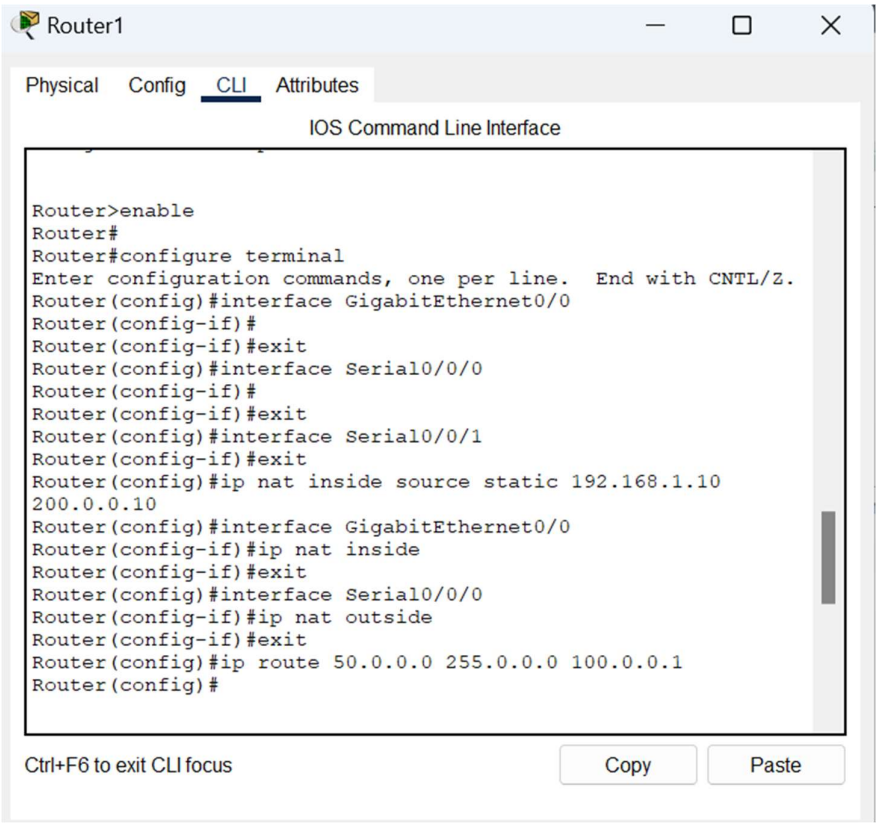
Static NAT Confiruration:

Router0:



EXPERIMENT NO:12 **Implementation of Network Address Translation**

Router1:



Output In Command Prompt:

Cisco Packet Tracer PC Command Line 1.0
C:\>ipconfig
FastEthernet0 Connection:(default port)

Connection-specific DNS Suffix..:
Link-local IPv6 Address.....: FE80::20C:85FF:FEE8:65D
IPv6 Address.....: ::
IPv4 Address.....: 10.0.0.10
Subnet Mask.....: 255.0.0.0
Default Gateway.....: ::
10.0.0.1

Bluetooth Connection:

Connection-specific DNS Suffix..:
Link-local IPv6 Address.....: ::
IPv6 Address.....: ::
IPv4 Address.....: 0.0.0.0
Subnet Mask.....: 0.0.0.0
Default Gateway.....: ::
0.0.0.0

C:\>ping 200.0.0.10

Pinging 200.0.0.10 with 32 bytes of data:

Reply from 200.0.0.10: bytes=32 time=9ms TTL=126
Reply from 200.0.0.10: bytes=32 time=1ms TTL=126
Reply from 200.0.0.10: bytes=32 time=1ms TTL=126
Reply from 200.0.0.10: bytes=32 time=1ms TTL=126

C:\>ping 192.168.1.10

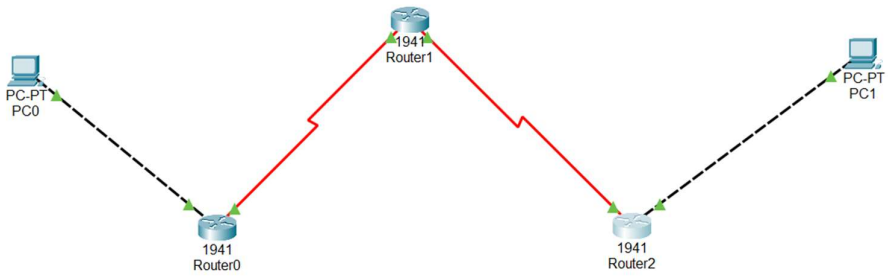
Pinging 192.168.1.10 with 32 bytes of data:

Reply from 10.0.0.1: Destination host unreachable.
Reply from 10.0.0.1: Destination host unreachable.
Reply from 10.0.0.1: Destination host unreachable.
Reply from 10.0.0.1: Destination host unreachable.

Ping statistics for 192.168.1.10:
Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),

EXPERIMENT NO:13 IMPLEMENTATION OF VPN

Cisco Packet Tracer:



Router0:

Router>en
Router#ping 172.18.1.2

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 172.18.1.2, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 3/4/6 ms

Router2:

Router>en
Router#ping 172.18.1.1

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 172.18.1.1, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 2/4/6 ms

Testing VPN Tunnel:

Router0:

```
Router#show interfaces Tunnel 400
Tunnel400 is up, line protocol is up (connected)
Hardware is Tunnel
Internet address is 172.18.1.2/16
MTU 17916 bytes, BW 100 Kbit/sec, DLY 50000 usec,
reliability 255/255, txload 1/255, rxload 1/255
Encapsulation TUNNEL, loopback not set
Keepalive not set
Tunnel source 10.0.0.1 (Serial0/0/0), destination 20.0.0.1
Tunnel protocol/transport GRE/IP
Key disabled, sequencing disabled
Checksumming of packets disabled
Tunnel TTL 255
Fast tunneling enabled
Tunnel transport MTU 1476 bytes
Tunnel transmit bandwidth 8000 (kbps)
Tunnel receive bandwidth 8000 (kbps)
Last input never, output never, output hang never
Last clearing of "show interface" counters never
Input queue: 0/75/0/0 (size/max/drops/flushes); Total output drops: 1
Queueing strategy: fifo
Output queue: 0/0 (size/max)
5 minute input rate 0 bits/sec, 0 packets/sec
--More--
```

Router2:

```
Router#show interfaces Tunnel 200
Tunnel200 is up, line protocol is up (connected)
Hardware is Tunnel
Internet address is 172.18.1.1/16
MTU 17916 bytes, BW 100 Kbit/sec, DLY 50000 usec,
reliability 255/255, txload 1/255, rxload 1/255
Encapsulation TUNNEL, loopback not set
Keepalive not set
Tunnel source 20.0.0.1 (Serial0/1/0), destination 10.0.0.1
Tunnel protocol/transport GRE/IP
Key disabled, sequencing disabled
Checksumming of packets disabled
Tunnel TTL 255
Fast tunneling enabled
Tunnel transport MTU 1476 bytes
Tunnel transmit bandwidth 8000 (kbps)
Tunnel receive bandwidth 8000 (kbps)
Last input never, output never, output hang never
Last clearing of "show interface" counters never
```

Input queue: 0/75/0/0 (size/max/drops/flushes); Total output drops: 1
Queueing strategy: fifo
Output queue: 0/0 (size/max)
5 minute input rate 0 bits/sec, 0 packets/sec
--More--

Tracing VPN tunnel:

C:\>tracert 192.168.2.2

Tracing route to 192.168.2.2 over a maximum of 30 hops:

1 0 ms 0 ms 0 ms 192.168.1.1
2 10 ms 11 ms 9 ms 172.18.1.1
3 6 ms 2 ms 2 ms 192.168.2.2

Trace complete.

Experiment no:14 **Communication Using HDLC**

Cisco Packet Tracer:

Show Interface for HDLC:

Router#show int se0/3/0
Serial0/3/0 is up, line protocol is up (connected)
Hardware is HD64570
Internet address is 192.168.1.2/30
MTU 1500 bytes, BW 1544 Kbit, DLY 20000 usec,
reliability 255/255, txload 1/255, rxload 1/255
Encapsulation HDLC, loopback not set, keepalive set (10 sec)
Last input never, output never, output hang never
Last clearing of "show interface" counters never
Input queue: 0/75/0 (size/max/drops); Total output drops: 0
Queueing strategy: weighted fair
Output queue: 0/1000/64/0 (size/max total/threshold/drops)
Conversations 0/0/256 (active/max active/max total)
Reserved Conversations 0/0 (allocated/max allocated)
Available Bandwidth 1158 kilobits/sec
5 minute input rate 0 bits/sec, 0 packets/sec
5 minute output rate 0 bits/sec, 0 packets/sec
0 packets input, 0 bytes, 0 no buffer
Received 0 broadcasts, 0 runts, 0 giants, 0 throttles
0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored, 0 abort
0 packets output, 0 bytes, 0 underruns
0 output errors, 0 collisions, 0 interface resets
0 output buffer failures, 0 output buffers swapped out
0 carrier transitions
DCD=up DSR=up DTR=up RTS=up CTS=up

Checking Connection By Pinging:

Router0:

Router#
%SYS-5-CONFIG_I: Configured from console by console
ping 192.168.1.1

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.1.1, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/10/19 ms

Router1:

Router#
%SYS-5-CONFIG_I: Configured from console by console
ping 192.168.1.2

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.1.2, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 5/8/11 ms

Experiment no:15 **Communication Using PPP**

Cisco Packet Tracer:

Show Interface for PPP:

Router>en
Router#show int se0/3/0
Serial0/3/0 is up, line protocol is up (connected)
Hardware is HD64570
Internet address is 192.168.1.2/30
MTU 1500 bytes, BW 1544 Kbit, DLY 20000 usec,
reliability 255/255, txload 1/255, rxload 1/255
Encapsulation PPP, loopback not set, keepalive set (10 sec)
LCP Open
Open: IPCP, CDPCP
Last input never, output never, output hang never
Last clearing of "show interface" counters never
Input queue: 0/75/0 (size/max/drops); Total output drops: 0
Queueing strategy: weighted fair
Output queue: 0/1000/64/0 (size/max total/threshold/drops)
Conversations 0/0/256 (active/max active/max total)
Reserved Conversations 0/0 (allocated/max allocated)
Available Bandwidth 1158 kilobits/sec
5 minute input rate 0 bits/sec, 0 packets/sec
5 minute output rate 0 bits/sec, 0 packets/sec
0 packets input, 0 bytes, 0 no buffer
Received 0 broadcasts, 0 runts, 0 giants, 0 throttles
0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored, 0 abort
0 packets output, 0 bytes, 0 underruns
0 output errors, 0 collisions, 0 interface resets
0 output buffer failures, 0 output buffers swapped out
0 carrier transitions
DCD=up DSR=up DTR=up RTS=up CTS=up

Checking Connection By Pinging:

Router0:

Router#ping 192.168.1.1

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.1.1, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 3/7/12 ms

Router1:

Router#
%SYS-5-CONFIG_I: Configured from console by console
ping 192.168.1.2

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.1.2, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 5/8/12 ms

MINI PROJECT

Secure File Sharing System

A COURSE PROJECT REPORT

By

Paritosh Sahu (RA2011031010012)
Vishal Aravinth A(RA2011031010013)
Shiva Kumar (RA2011031010017)
Aakash G (RA2011031010015)

Under the guidance of

Thenmalar.S

In partial fulfilment for the Course

of

18CSC302J - COMPUTER NETWORKS

in Networking and communication

FACULTY OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

Kattankulathur, Chenpalpattu District

NOVEMBER 2022

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that this mini project report "**File Transfer Protocol with Encryttion** " is the bonafide work of **Paritosh Sahu (RA2011031010012)**, **Vishal Aravinth (RA2011031010013),Shiva Kumar.R (RA2011031010017) and Aakash.G (RA2011031010015)** who carried out the project work under my supervision.

SIGNATURE

Thenmalar.S
Assistant Professor
Department of of Networking and Communication
SRM Institute of Science and Technology

ACKNOWLEDGEMENT

We express our heartfelt thanks to our honorable **Vice Chancellor Dr. C. MUTHAMIZHCHELVAN**, for being the beacon in all our endeavors.

We would like to express my warmth of gratitude to our **Registrar Dr. S. Ponnusamy,** for his encouragement

We express our profound gratitude to our **Dean (College of Engineering and Technology) Dr. T. V.Gopal,** for bringing out novelty in all executions.

We would like to express my heartfelt thanks to Chairperson, School of Computing **Dr. Revathi Venkataraman,** for imparting confidence to complete my course project

We wish to express my sincere thanks to **Course Audit Professor Dr.Annapurani Panaiyappan, Professor and Head, Department of Networking and Communications** and **Course Coordinators** for their constant encouragement and support.

We are highly thankful to our my Course project Faculty **Thenmalar.S , Assistant professor, Department of Networking and Communications** for his/her assistance, timely suggestion and guidance throughout the duration of this course project.

We extend my gratitude to our **HoD Dr.Annapurani Panaiyappan, Professor and Head, Department of Networking and Communications** and my Departmental colleagues for their Support.

Finally, we thank our parents and friends near and dear ones who directly and indirectly contributed to the successful completion of our project. Above all, I thank the almighty for showering his blessings on me to complete my Course project.

TABLE OF CONTENTS

ABSTRACT

The main objective of our project "Modelling a prototype for ClientServer based File Sharing System"is to develop a high performance network, whose primary purpose is to build a storage devices which stores files and allow users to access, share and modify it from anywhere. Generally users create files and save that on their hard drives which consumes their memory and if they want to share that file with any other user then personally they have to send their files using some services which is tedious work. The other factor is security issue i.e. while sharing the file, user is not sure that whether it reaches to the correct destination without tempering file. If user wanted to modify the file which it had saved on its hard drive, then either he has to access same computer or has to take manually that file in any other hard

drive.

1. INTRODUCTION

"Building a secure model for clientserver based file sharing system."

"Modelling a prototype for ClientServer based File Sharing System" will allow us to have good understanding for the clientserver model. Multithreading which is necessary model for any Server will be studied in detail. File creation on server side after it is uploaded by user will also be understood. After requesting for the file, how the file is deployed on respective client machine is also discussed. File Sharing between users and giving access rights to another user is another aspect which is discussed. and User Authentication using different Cryptographic algorithm is also implemented. Memory management is also important aspect which will be covered in project.

Following aspects of the prototype model can be modernized or optimised in terms of memory
optimization, user perception and scalability:

1. Dynamic memory allocation on server: Dynamic memory allocation on server allows us to optimally use the unused user space when assigned statically. That is it can be managed between more users as a result the 10GB space can be allocated to more than 10 users which is the user limiting case when assigned statically like 1GB per user.
2. Scalability when users requests for more memory than the room allocated to him/her, they can purchase it from the server memory or from its peers at comparatively low cost by using safe transaction method.

2. LITERATURE SURVEY

The main focus of this paper is to model a prototype of clientserver based file sharing system. The earliest design uses a central server (or server cluster) to coordinate participating nodes and to maintain an index of all available files being shared. When a peer node joins the system, it contacts the central server and sends a list of the local files that are available for other peers to download (shared files). To locate a file, a peer sends a query to the central server, which performs a database lookup and responds with a list of peers that have the desired file. If a peer leaves the system, its list of shared files is removed from the central server. We will denote such architecture as a CIA (Centralized Indexing Architecture). Broadly the paper can be divided into 3 major modules. Mainly, User authentication, file organization and sharing and Multiuser request handling. Let us look at each of these modules in discrete and the technical background associated to it.

Firstly, User authentication is a means of identifying the user and verifying that the user is allowed to access some restricted service. That is when user logs in to the server, you verify the authenticity of the server whether he is allowed to access the server resources or not. Once user gets through the authentication phase he gets the full privilege to access his files on the server. Public key cryptography is based on very complex mathematical problems that require very specialized knowledge. Public key School of Engineering cryptography makes use of two keys, one private and the other public. The two keys are linked together by way of an extremely complex mathematical equation. The private key is used to decrypt and also to encrypt messages between the communicating machines. Both encryption and verification of signature is accomplished with the public key.

Secondly, the major functionality of our project is implemented in file organization and sharing module. Network file sharing is the process of copying files from one computer to another using a live network connection. This paragraph describes the different methods and networking technologies available to help you share files. Let's begin with FTP File Transfers, File Transfer Protocol (FTP) is an older but still popular method to share files on the Internet. A central computer called the FTP server holds all the files to be shared, while remote computers running FTP client software can log in to the server to obtain copies.

In order to facilitate multiple clients accessing server at the same time, the third module focuses on handling multiclient requests and associated conflicts in accessing the server resources. This can be implemented using multithread concept of operating system. In a multithreaded process on a single processor, the processor can switch execution resources between threads, resulting in concurrent execution. In the same multithreaded process in a sharedmemory multiprocessor environment, each thread in the process can run on a separate processor at the same time, resulting in parallel execution.

3. METHODOLOGY

We are designing and implementing a small model of file sharing system. This system uses a client server architecture to make this thing happen and on top of clientserver architecture we are using some concepts of file system to allocate some storage to users. Here mainly we have 6 tasks, these tasks are performed by the client and server to provide services to user.

Tasks are as follows:

1. Users can register to get the benefits of file sharing services offered by file sharing system.
2. Server will always does user authentication.
3. Users can upload files on the file sharing system (server).
4. User can download the file from the file sharing system(server).
5. Users can share their file with other registered users.
6. Users can delete the file from the file sharing system.

Workflow of all the above tasks:

In the workflow, we are going to explain briefly about all the steps our file sharing system will take in order to serve the requesting service.

1. Registration Workflow:

● User sends registration request with register command, and gives username and password as the parameters of the register command, using file sharing client (client performs encryption on password before sending using the shared key between client and server) to file sharing system (file sharing server).

For example “register abc”.

● File sharing system (server) receives the registration request and makes an entry of this username in the database with the encrypted form of the password (for the security purpose), makes a directory named “userId” in the file system allocates some storage (in our case 1GB) to this user.

2. User Authentication Workflow:

● Before allowing any user to use the services of the file sharing system, system (server) first validates that the user requesting for the service is a registered user (valid) or not. To do that server follows the following steps:

● Receives username and the password and performs a search in the user information database to find an entry of the user with the given username.

● If server finds an entry in the database for the requesting user it further proceeds for the the password verification otherwise it sends a message to user that says username is not valid. Server gives maximum three attempts to user to reenter the username, if user fails to provide correct username in all the attempts then after 3rd attempt server immediately drops the connection.

● if server finds that the password provided by the requesting user is correct then only it allows user to use the services otherwise sends a message to user that password doesn't match. Server gives maximum three attempts to user to reenter
the password, if user fails to provide correct password in all the attempts then after 3rd attempt server immediately drops the connection

3. File Upload Workflow

● User sends file upload request with upload command, and gives filename (file path) as the parameter of the upload command, using file sharing client to file sharing system (file sharing server).
For example "upload /home/rajni/xyz.txt".

● Server receives file upload request and checks that service requesting user has enough space available or not on the basis of the file size user wants to upload.

● If server finds that user has enough available space it puts user's file named "xyz.txt" in the user's directory named as "userId" of this user. Otherwise server sends a message to user that you don't have enough storage available to upload the file "xyz.txt".

- After uploading the file server updates the available storage of the user by subtracting the size of the uploaded file from the current available storage of the user.

4. File Download WorkFlow:

● User sends file download request with download command, and gives filename and a path where user wants to save file in its local machine as the parameters of the download command, using file sharing client to file sharing system (file sharing server).
For example “download xyz.txt /home/rajni/”.

● Server receives the download request from the user and checks in the user’s directory named as “userId” that the requesting file (“xyz.txt”) for downloading resides in the requesting user’s directory or not.

● If server finds the requesting file in the user’s directory, it sends the file to the requesting client and then client saves the received file in the place specified by the user as the second parameter of the command.

5. Delete File WorkFlow:

● User sends file delete request with delete command, and gives filename as the parameter of the upload command, using file sharing client to file sharing system (file sharing server).
For example: delete xyz.txt

● Server receives the delete request from the user and checks in the user’s directory named as “userId” that the requesting file (“xyz.txt”) to delete resides in the requesting user’s directory or not.

● If server finds the file in the user’s directory then it removes the entry of the file from the user’s directory and updates the current available space of user by doing addition of the file size in current available space. Otherwise sends a message to user that file you are requesting to delete doesn’t exist in your directory.

4. REQUIREMENTS

4.1 Requirement Analysis-

From the given scenario, we draw the following requirements:

1. Identifying the appropriate hardware which would be used (Cisco Packet Tracer)
2. Users on the internet should have access only to the public IP address of the server and not the private IP address.
3. The users in the organization should have full access to the server.
4. TCP/IP Network design with IP addressing
5. Features and configuration required on the hardware with explanation

We need to configure a network design keeping the following requirements in mind.

4.2 Hardware Requirement

From the given scenario, we draw the following requirements:

Hardware Required:

1x Server – PT Primary Server

1x Router (For address 10.0.0.1)

1x Switch:

1x Primary Switch

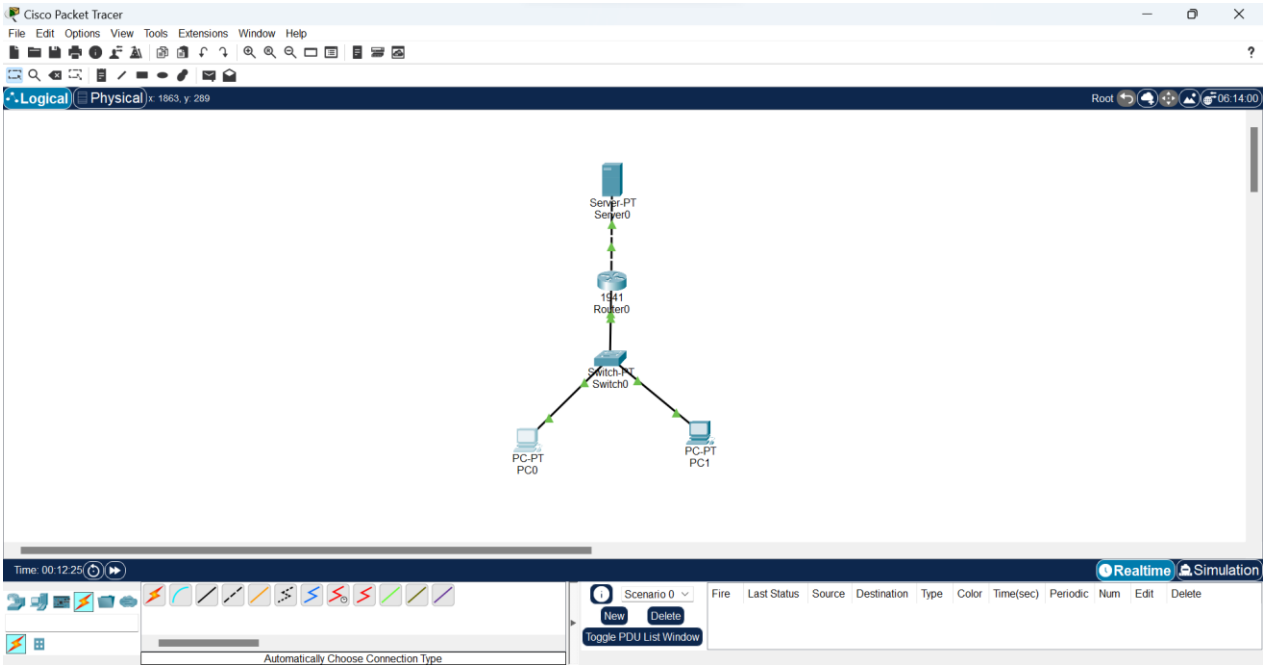
2x End Devices:

2x PCs for File uploading and downloading

5. ARCHITECTURE AND DESIGN

5.1 Network Architecture

The network architecture is as follows:



The architecture consists of :

- Server
- Router
- Switch
- PCs

6. IMPLEMENTATION

6.1 Address Table

The address table is as follows:

Device	Interface	Address
Server	Fa0	10.10.10.2
Router	Fa0/0	10.0.0.1
	Fa0/1	192.168.0.1
PC0	Fa0/0	192.168.0.2
PC1	Fa0/0	192.168.0.3

Static Routing is used on all the routers to interconnect the networks.

7. CODE

Network Code:

```
#!/usr/bin/env python3
#network.py

import os, sys, getopt, time

NET_PATH = './'
ADDR_SPACE = 'ABC'
CLEAN = False
TIMEOUT = 0.500  # 500 millisec

def read_msg(src):
	global last_read

	out_dir = NET_PATH + src + '/OUT'
	msgs = sorted(os.listdir(out_dir))

	if len(msgs) - 1 <= last_read[src]: return '', ''

	next_msg = msgs[last_read[src] + 1]
	dsts = next_msg.split('--')[1]
	with open(out_dir + '/' + next_msg, 'rb') as f: msg = f.read()

	last_read[src] += 1
	return msg, dsts

def write_msg(dst, msg):
	in_dir = NET_PATH + dst + '/IN'
	msgs = sorted(os.listdir(in_dir))

	if len(msgs) > 0:
		last_msg = msgs[-1]
		next_msg = (int.from_bytes(bytes.fromhex(last_msg), byteorder='big') +
1).to_bytes(2, byteorder='big').hex()
	else:
		next_msg = '0000'

	with open(in_dir + '/' + next_msg, 'wb') as f: f.write(msg)

	return

# ------------
# main program
# ------------
```

```
try:
	opts, args = getopt.getopt(sys.argv[1:], shortopts='hp:a:c', longopts=['help', 'path=',
'addrspace=', 'clean'])
except getopt.GetoptError:
	print('Usage: python network.py -p <network path> -a <address space> [--clean]')
	sys.exit(1)

#if len(opts) == 0:
#	print('Usage: python network.py -p <network path> -a <address space> [--clean]')
#	sys.exit(1)

for opt, arg in opts:
	if opt == '-h' or opt == '--help':
		print('Usage: python network.py -p <network path> -a <address space> [--
clean]')
		sys.exit(0)
	elif opt == '-p' or opt == '--path':
		NET_PATH = arg
	elif opt == '-a' or opt == '--addrspace':
		ADDR_SPACE = arg
	elif opt == '-c' or opt == '--clean':
		CLEAN = True

ADDR_SPACE = ''.join(sorted(set(ADDR_SPACE)))

if len(ADDR_SPACE) < 2:
	print('Error: Address space must contain at least 2 addresses.')
	sys.exit(1)

for addr in ADDR_SPACE:
	if addr not in 'ABCDEFGHIJKLMNOPQRSTUVWXYZ':
		print('Error: Addresses must be capital letters from the 26-element English
alphabet.')
		sys.exit(1)

if (NET_PATH[-1] != '/') and (NET_PATH[-1] != '\\'): NET_PATH += '/'

if not os.access(NET_PATH, os.F_OK):
	print('Error: Cannot access path ' + NET_PATH)
	sys.exit(1)

print('----------------------------------------')
print('Network is running with the following input:')
print('  Network path: ' + NET_PATH)
print('  Address space: ' + ADDR_SPACE)
print('  Clean-up requested: ', CLEAN)
print('----------------------------------------')
```

```
# create folders for addresses if needed
for addr in ADDR_SPACE:
        addr_dir = NET_PATH + addr
        if not os.path.exists(addr_dir):
                print('Folder for address ' + addr + ' does not exist. Trying to create it... ',
end='')
                os.mkdir(addr_dir)
                os.mkdir(addr_dir + '/IN')
                os.mkdir(addr_dir + '/OUT')
                print('Done.')

# if program was called with --clean, perform clean-up here
# go through the addr folders and delete messages
if CLEAN:
        for addr in ADDR_SPACE:
                in_dir = NET_PATH + addr + '/IN'
                for f in os.listdir(in_dir): os.remove(in_dir + '/' + f)
                out_dir = NET_PATH + addr + '/OUT'
                for f in os.listdir(out_dir): os.remove(out_dir + '/' + f)

# initialize state (needed for tracking last read messages from OUT dirs)
last_read = {}
for addr in ADDR_SPACE:
        out_dir = NET_PATH + addr + '/OUT'
        msgs = sorted(os.listdir(out_dir))
        last_read[addr] = len(msgs) - 1

# main loop
print('Main loop started, quit with pressing CTRL-C...')
while True:
        time.sleep(TIMEOUT)
        for src in ADDR_SPACE:
                msg, dsts = read_msg(src)                       # read outgoing message
                if dsts != '':
        # if read returned a message...
                        if dsts == '+': dsts = ADDR_SPACE
        # handle broadcast address +
                        for dst in dsts:
        # for all destinations of the message...
                                if dst in ADDR_SPACE:
        # destination must be a valid address
                                        write_msg(dst, msg)                     # write incoming
message
```

Server Code:

```
###-----------------------------------------###
# server operations for secure file transfer
###-----------------------------------------###

import os
import sys
import threading

from netinterface import network_interface
from server_interface import Serverif

NET_PATH = './network'
OWN_ADDR = 'S'
LOGGED_IN_USER = None
TIMEOUT_DELAY = 60.0

if (NET_PATH[-1] != '/') and (NET_PATH[-1] != '\\'): NET_PATH += '/'

if not os.access(NET_PATH, os.F_OK):
    print('Error: Cannot access path ' + NET_PATH)
    sys.exit(1)

if len(OWN_ADDR) > 1: OWN_ADDR = OWN_ADDR[0]

if OWN_ADDR not in network_interface.addr_space:
    print('Error: Invalid address ' + OWN_ADDR)
    sys.exit(1)

netif = network_interface(NET_PATH, OWN_ADDR)
serverif = Serverif(OWN_ADDR, NET_PATH)
# status, msg = netif.receive_msg(blocking=True)  # when returns, status is True and msg
contains a message

def timeout():
    print("Session Expired.")
    serverif.force_logout(netif)

print('Main loop started, quit with pressing CTRL-C...')
timer = threading.Timer(TIMEOUT_DELAY, timeout)
timer.start()
while True:
    # wait for message
    status, msg = netif.receive_msg(blocking=True)  # when returns, status is True and msg
contains a message
```

```
    serverif.process_msg(netif, status, msg)
    if status:
        timer.cancel()
        timer = threading.Timer(TIMEOUT_DELAY, timeout)
        timer.start()
        continue
```

Client Code:

```
###-----------------------------------------------###
# implementation of client operations for secure
# file transfer
###-----------------------------------------------###

import sys, os
from netinterface import network_interface
from client_interface import login, welcome, build_msg, process_input
from user import User
from aes_ops import check_sqn, decrypt

SUCCESS = '1'
FAILURE  = '0'
FORCED_LOGOUT = '2'

NET_PATH = './network'
OWN_ADDR = input('Enter user address: ')

if (NET_PATH[-1] != '/') and (NET_PATH[-1] != '\\'): NET_PATH += '/'

if not os.access(NET_PATH, os.F_OK):
    print('Error: Cannot access path ' + NET_PATH)
    sys.exit(1)

if len(OWN_ADDR) > 1: OWN_ADDR = OWN_ADDR[0]

if OWN_ADDR not in network_interface.addr_space:
    print('Error: Invalid address ' + OWN_ADDR)
    sys.exit(1)

netif = network_interface(NET_PATH, OWN_ADDR)
user = User(OWN_ADDR)
dst = 'S'    ## set destination to server

## login protocol
user.session = login(netif, OWN_ADDR)
if user.session is not None:
    welcome(user.addr)
```

```
    while True:
        # user.session.print()
        inp = input('Type a command: ')

        cmd, arg = process_input(inp)
        if cmd is None:
            continue

        # build message based on input
        msg = build_msg(user.addr, user.session, cmd, arg)
        if cmd == 'UPL' and msg is None:
            print('Invalid filename')
            continue

        # send message
        netif.send_msg(dst, msg)
        user.session.sqn_snd += 1

        # wait for response
        status, rsp = netif.receive_msg(blocking=True)  # when returns, status is True and
msg contains a message

        # check sqn_rsp > user.session.sqn_rcv (in header, don't need to decrypt)
        sqn_rsp = int.from_bytes(rsp[17:21], byteorder='big')
        if check_sqn(user.session.sqn_rcv, sqn_rsp):
            # decrypt rsp
            addr, rsp_code, arg = decrypt(rsp, user.session.key)  # we don't care about addr
(S) or arg (should be empty string)

            # set user.session.rsp = sqn_rsp
            user.session.sqn_rcv = sqn_rsp

            if rsp_code == FORCED_LOGOUT:
                print("Session Expired. You have been logged out.")
                quit()

            # check success/failure code
            if rsp_code == SUCCESS:
                if cmd == 'LOGOUT':
                    print('Logout success. Goodbye.')
                    quit()
                if arg is not None:
                    print(arg)
            else:
                print('Unable to complete command')
        # print success/failure message
        else:
            print('message sequence number not accepted')
```

8. RESULTS AND DISCUSSION

8.1 Connection Check

The network connections were checked by ping requests:

```
C:\>ping 10.10.10.2

Pinging 10.10.10.2 with 32 bytes of data:

Reply from 10.10.10.2: bytes=32 time<1ms TTL=127
Reply from 10.10.10.2: bytes=32 time<1ms TTL=127
Reply from 10.10.10.2: bytes=32 time<1ms TTL=127
Reply from 10.10.10.2: bytes=32 time<1ms TTL=127

Ping statistics for 10.10.10.2:
Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

8.2 Uploading file to Server

```
C:\>ftp 10.10.10.2
Trying to connect...10.10.10.2
Connected to 10.10.10.2
220- Welcome to PT Ftp server
Username:u
331- Username ok, need password
Password:
230- Logged in
(passive mode On)
ftp>put test.txt
```

8.3 Downloading File from Server

```
Cisco Packet Tracer PC Command Line 1.0
C:\>ftp 10.10.10.2
Trying to connect...10.10.10.2
Connected to 10.10.10.2
220- Welcome to PT Ftp server
Username:u
331- Username ok, need password
Password:
230- Logged in
(passive mode On)
ftp>get test.txt

Reading file test.txt from 10.10.10.2:
File transfer in progress...

    [Transfer complete - 34 bytes]
```

9. CONCLUSION AND FUTURE ENHANCEMENT

Overall this system makes users close to their data i.e. users can see, modify, and share their data anytime from anywhere, they don't have to carry their computer always with them to work on their data.

So here we are using a clientserver server architecture which makes this thing possible. A user who wants to use the services of this system must have a client of the system and that client sends request to an always running server then server serves the requesting service to the user.

There are main 6 commands in this filesharing system as follows:
1. Users can register to get the benefits of file sharing services offered by file sharing system.
2. Server will always does user authentication before allowing any user to use the services of the system.
3. After successful authentication, users can upload files on the file sharing system (server).
4. User can download the file from the file sharing system(server).
5. Users can share their file with other registered users.
6. Users can delete the file from the file sharing system.

Future scope:

This project is a small model of the filesharing area applications. But there are many future scopes for this project some of them are as follows:

- There can be a very sophisticated client with very intuitive graphical user interface. If time permits we will try to do this in this project itself.
- For removing the need of desktop based client we can think about a webbased client. That will provide more flexibility to users to use the filesharing system.
- we can make our server more robust by doing some kind of replication of the server's data and by running more than one server. Since in this project we are using only one centralized server if something wrong happens users will lose their data and that will be a very big loss.

REFERENCES

https://www.geeksforgeeks.org/encrypt-and-decrypt-files-using-python/

https://www.tutorialspoint.com/cryptography_with_python/cryptography_with_python_quick_guide.htm

https://w3guides.com/tutorial/file-transfer-protocol-server-configuration-using-cisco-packet-tracer

AWS BADGE AND CERTIFICATE


aws
ACADEMY
Cloud Security
Foundations

aws academy

Vishal Aravinth A

Certificate of Completion for

AWS Academy Graduate - AWS Academy Cloud Security Foundations

Course hours completed

20 hours

Issued on

11/10/2022

Digital badge

https://www.credly.com/go/GCpv1QUg