

SSN COLLEGE OF ENGINEERING (Autonomous)
(Affiliated to Anna University, Chennai)
DEPARTMENT OF CSE

UCS 1511 NETWORK LABORATORY

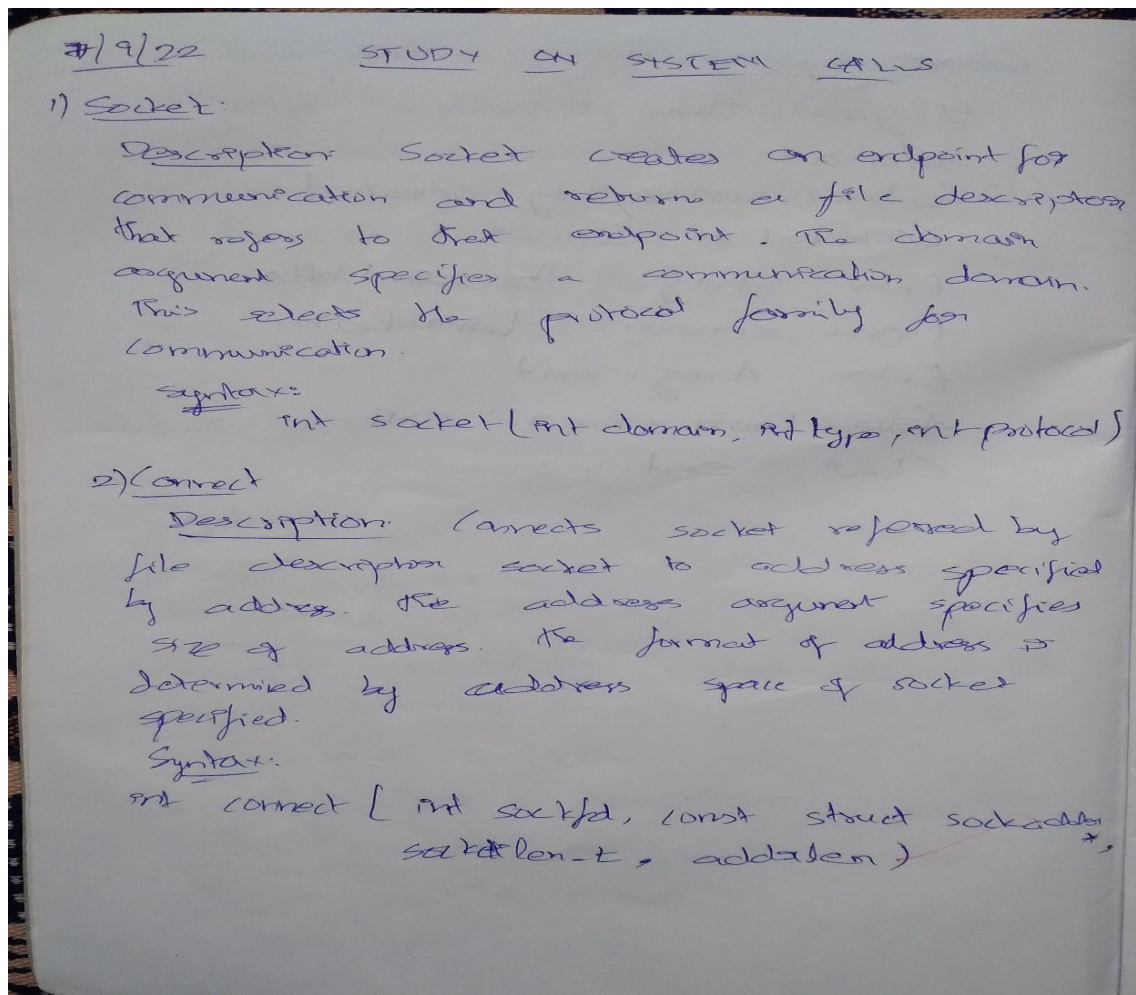
Faculty In-charge: Dr.S.V.Jansi Rani, Associate Professor / CSE

Batch: 2020-2024 Year: III Sem: V Sec: B

=====

Assignment 1 : System Commands

Objective: To be proficient in executing networking system commands.



3) Bind:

Description: Bind assigns the address specified by `addr` to socket referred to by `fd` `sockfd`.

Syntax:

```
int bind (int sockfd, const struct sockaddr *,  
          socklen_t addrlen);
```

4) Listen

Description:

It marks the socket referred to by `sockfd` as a passive socket, that is a socket that will be used to accept incoming connection using `accept`.

Syntax:

```
int listen (int sockfd, int backlog);
```

5) Accept:

Description: used with connection based socket types. It extracts the 1st connection request in the queue of pending connections request on the ~~queue~~ socket.

Syntax: `int accept (int sockfd, struct sockaddr *,
socklen_t *restrict, addr_len)`

6) Close

Description: It closes a file description, so that it no longer refers to any file and may be reused.

Syntax: `int close (int fd)`

7) Read

Description: It attempts to read upto `count` bytes from file descriptor `fd` into buffer starting at `buf`.

Syntax:

`ssize_t read (int fd, void *buf, ssize_t count)`

8) Write

Description: It writes upto `count` bytes from buffer starting at `buf` to the file referred to by `fd`.

Syntax: `ssize_t write (int fd, const void *buf, ssize_t count)`

9) Send:

Description: Used to send a message to another socket. It is used only socket in a connected state.

Syntax

```
ssize_t send(int sockfd, const void * buf,  
             ssize_t len, int flags)
```

flags : MSG - CONFIRM
 MSG - DONTROUTE
 MSG - EOR etc

10) recv:

Description: Used to receive messages from a socket, on both connection less and connection-oriented sockets.

Syntax

```
ssize_t recv(int sockfd, void * buf, size_t  
            len, int flags)
```

11) Send to:

Description: Transmitted message to socket in a connected state. The message is found in buf and length len.

Syntax:

```
ssize_t sendto(int sockfd, const void *buf,  
size_t len, int flags, const struct  
sockaddr *dest_addr, socklen_t  
addrlen)
```

12) recv-from.

Description: Used to get message from a socket on both connection-less and connection-oriented sockets. Returns length of message on successful completion.

Syntax:

```
ssize_t recv-from(int sockfd, void *  
restrict buf, size_t len, int flags,  
struct sockaddr * restrict addr,  
socklen_t, * restrict addrlen);
```

Assignment 2 : Simple Client server Application

Objective:

To write a program to establish a connection between a server and a client and exchange a message

Algorithm:

Server:

1. Create a socket address of type `sockaddr_in` for server and define its family as `AF_INET`, port number and IP address
2. Create a new file descriptor for the socket (used for communication between server and clients) using `socket()` system call. Return error if socket creation fails
3. Bind the server socket address to this socket using `bind()` system call. Return error if binding fails
4. Listen to any incoming connection request using the `listen()` system call. Return error if listening fails
5. Accept the first pending request from a client using `accept()` system call and get the client's socket file descriptor. Print error if there are no pending requests
6. Receive a message from the client using `recv()` system call.
 - 6.1 Print error if receiving message fails
7. Close the socket file descriptor

Client:

1. Create a socket address of type `sockaddr_in` for client and define its family as `AF_INET`, port number and IP address. Server and client must share the port number

2. Create a new file descriptor for the socket used for communication between server and the client using `socket()` system call. Return error if socket creation fails
3. Connect to the server socket address using `connect()` system call. Return error if connection fails
4. Write a message to the server using `write()` system call. If the input in terminal is '.' Then terminate the program.
5. Wait in a while loop for the server to reply.
6. Using the `recv()` system call , read the message from the server and go to step 4.
7. Close the socket file descriptor

Code:

Server.c:

```
#include <netinet/in.h> //structure for storing address information
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h> //for socket APIs
#include <sys/types.h>

int main(int argc, char const* argv[])
{

    // create server socket similar to what was done in
    // client program
    int servSockD = socket(AF_INET, SOCK_STREAM, 0);

    // string store data to send to client
    char serMsg[255];
    scanf("%[^\\n]%*c", serMsg);

    // define server address
    struct sockaddr_in servAddr;

    servAddr.sin_family = AF_INET;
    servAddr.sin_port = htons(9001);
    servAddr.sin_addr.s_addr = INADDR_ANY;
```

```

// bind socket to the specified IP and port
bind(servSockD, (struct sockaddr*)&servAddr,
sizeof(servAddr));

// listen for connections
listen(servSockD, 1);

// integer to hold client socket.
int clientSocket = accept(servSockD, NULL, NULL);

// send's messages to client socket
send(clientSocket, serMsg, sizeof(serMsg), 0);

return 0;
}

```

Client.c:

```

#include <netinet/in.h> //structure for storing address information
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h> //for socket APIs
#include <sys/types.h>

int main(int argc, char const* argv[])
{
    int sockD = socket(AF_INET, SOCK_STREAM, 0);

    struct sockaddr_in servAddr;

    servAddr.sin_family = AF_INET;
    servAddr.sin_port
    = htons(9001); // use some unused port number
    servAddr.sin_addr.s_addr = INADDR_ANY;

    int connectStatus
    = connect(sockD, (struct sockaddr*)&servAddr,
        sizeof(servAddr));

    if (connectStatus == -1) {
        printf("Error...\n");
    }
}

```



```

    }

    else {
        char strData[255];

        recv(sockD, strData, sizeof(strData), 0);
        printf("Message: %s\n",strData);

    }

    return 0;
}

```

Sample I/O:

```

root@spl14:~/Documents/205001098# gcc client.c
root@spl14:~/Documents/205001098# ./a.out
hello

```

```

root@spl14:~/Documents/205001098# gcc server.c
root@spl14:~/Documents/205001098#
root@spl14:~/Documents/205001098#
root@spl14:~/Documents/205001098# ./a.out
Message: hello
root@spl14:~/Documents/205001098# clear

```

Assignment-3: Echo Server

Objective:

To write a program to establish connection between a server and a client and to send an echo message for the message sent by the client from the server.

Algorithm:

Server:

1. Create a socket address of type `sockaddr_in` for server and define its family as `AF_INET`, port number and IP address
2. Create a new file descriptor for the socket (used for communication between server and clients) using `socket()` system call. Return error if socket creation fails
3. Bind the server socket address to this socket using `bind()` system call. Return error if binding fails
4. Listen to any incoming connection request using `listen()` system call. Return error if listening fails
5. Accept the first pending request from a client using `accept()` system call and get the client's socket file descriptor. Print error if there are no pending requests
6. Receive a message from the client using `recv()` system call and store it on some buffer.
 - a. Print error if receiving message fails
7. Then using the `write()` system call , reply with same message that is stored in the buffer back to the client.
8. Close the socket file descriptor

Client:

1. Create a socket address of type `sockaddr_in` for client and define its family as `AF_INET`, port number and IP address. Server and client must share the port number
2. Create a new file descriptor for the socket used for communication between server and the client using `socket()` system call. Return error if socket creation fails
3. Connect to the server socket address using `connect()` system call. Return error if connection fails
4. Write a message to the server using `write()` system call. If the input in terminal is '.' Then terminate the program.
5. Wait in a while loop for the server to reply.
6. Using the `recv()` system call , read the message from the server and go to step 4.
7. Close the socket file descriptor

Code:

Server:

```
#include <netinet/in.h> //structure for storing address information
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h> //for socket APIs
#include <sys/types.h>

int main(int argc, char const* argv[])
{
    // create server socket similar to what was done in
    // client program
    int servSockD = socket(AF_INET, SOCK_STREAM, 0);

    // string store data to send to client
    char serMsg[255];
    scanf("%[^\n]*c", serMsg);

    // define server address
    struct sockaddr_in servAddr;
```



```

servAddr.sin_family = AF_INET;
servAddr.sin_port = htons(9001);
servAddr.sin_addr.s_addr = INADDR_ANY;

// bind socket to the specified IP and port
bind(servSockD, (struct sockaddr*)&servAddr,
sizeof(servAddr));

// listen for connections
listen(servSockD, 1);

// integer to hold client socket.
int clientSocket = accept(servSockD, NULL, NULL);

// send's messages to client socket
send(clientSocket, serMsg, sizeof(serMsg), 0);
char echoData[255];
recv(clientSocket,echoData,sizeof(echoData),0);
printf("Echo:%s",echoData);
return 0;
}

```

Client:

```

#include <netinet/in.h> //structure for storing address information
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h> //for socket APIs
#include <sys/types.h>

int main(int argc, char const* argv[])
{
    int sockD = socket(AF_INET, SOCK_STREAM, 0);

    struct sockaddr_in servAddr;

    servAddr.sin_family = AF_INET;
    servAddr.sin_port
= htons(9001); // use some unused port number
    servAddr.sin_addr.s_addr = INADDR_ANY;

    int connectStatus

```

```

    = connect(sockD, (struct sockaddr*)&servAddr,
              sizeof(servAddr));

    if (connectStatus == -1) {
        printf("Error...\n");
    }

    else {
        char strData[255];

        recv(sockD, strData, sizeof(strData), 0);
        printf("Message: %s\n",strData);
        printf("Sending Echo to server...\n");
        send(sockD,strData,sizeof(strData),0);
    }

    return 0;
}

```

Sample I/O:

```

root@spl14:~/Documents/205001098# gcc client2.c
root@spl14:~/Documents/205001098# ./a.out
hiii
message back : hiii
root@spl14:~/Documents/205001098# █

```

```

root@spl14:~/Documents/205001098# gcc server2.c
root@spl14:~/Documents/205001098# ./a.out
Message: hiii
root@spl14:~/Documents/205001098# █

```

Assignment-4: Simple Client Server Chat Application

Objective:

Write a program to exchange messages between server and client programs.

Algorithm:

Server:

1. Create a socket address of type `sockaddr_in` for server and define its family as `AF_INET`, port number and IP address
2. Create a new file descriptor for the socket (used for communication between server and clients) using `socket()` system call. Return error if socket creation fails
3. Bind the server socket address to this socket using `bind()` system call. Return error if binding fails
4. Listen to any incoming connection request using `listen()` system call. Return error if listening fails
5. Accept the first pending request from a client using `accept()` system call and get the client's socket file descriptor. Print error if there are no pending requests
6. Receive a message from the client using `recv()` system call.
 - 6.1 Print error if receiving message fails
7. Then using the `write()` system call , reply with some message to the client.
8. Close the socket file descriptor

Client:

1. Create a socket address of type `sockaddr_in` for client and define its family as `AF_INET`, port number and IP address. Server and client must share the port number
2. Create a new file descriptor for the socket used for communication between server and the client using `socket()` system call. Return error if socket creation fails
3. Connect to the server socket address using `connect()` system call. Return error if connection fails
4. Write a message to the server using `write()` system call. If the input in terminal is '.' Then terminate the program.
5. Wait in a while loop for the server to reply.
6. Using the `recv()` system call , read the message from the server and go to step 4.
7. Close the socket file descriptor

Code:

Server:

```
#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 8080
#define SA struct sockaddr

// Function designed for chat between client and server.
void func(int connfd)
{
    char buff[MAX];
    int n;
    // infinite loop for chat
    for (;;) {
        bzero(buff, MAX);
```

```

// read the message from client and copy it in buffer
read(connfd, buff, sizeof(buff));
// print buffer which contains the client contents
printf("From client: %s\t To client : ", buff);
bzero(buff, MAX);
n = 0;
// copy server message in the buffer
while ((buff[n++] = getchar()) != '\n')
    ;

// and send that buffer to client
write(connfd, buff, sizeof(buff));

// if msg contains "Exit" then server exit and chat ended.
if (strncmp("exit", buff, 4) == 0) {
    printf("Server Exit...\n");
    break;
}
}
}

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

    // socket create and verification
    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));

    // assign IP, PORT
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servaddr.sin_port = htons(PORT);

```

```

// Binding newly created socket to given IP and verification
if ((bind(sockfd, (SA*)&servaddr, sizeof(servaddr))) != 0) {
    printf("socket bind failed...\n");
    exit(0);
}
else
    printf("Socket successfully binded..\n");

// Now server is ready to listen and verification
if ((listen(sockfd, 5)) != 0) {
    printf("Listen failed...\n");
    exit(0);
}
else
    printf("Server listening..\n");
len = sizeof(cli);

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

// Function for chatting between client and server
func(connfd);

// After chatting close the socket
close(sockfd);
}

```

Client:

```

#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#define MAX 80

```



```

#define PORT 8080
#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 ((strcmp(buff, "exit", 4)) == 0) {
            printf("Client Exit...\n");
            break;
        }
    }
}

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

    // socket create and verification
    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));

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

```

```

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

// function for chat
func(sockfd);

// close the socket
close(sockfd);
}

```

Sample I/O:

```

    close(sockfd);
root@spl14:~/Documents/chat2# ./a.out
Socket successfully created..
connected to the server..
Enter the string : hiiiii
From Server : byeeee
Enter the string : exit
From Server : exit
Client Exit...
root@spl14:~/Documents/chat2#

```

```
root@spl13: ~/205001095/ex5
root@spl13:~/205001095/ex5# gcc twoserver.c
twoserver.c: In function 'func':
twoserver.c:23:3: warning: implicit declaration of function 'read' [-Wimplicit-function-declaration]
  read(connfd, buff, sizeof(buff));
  ^
twoserver.c:35:3: warning: implicit declaration of function 'write' [-Wimplicit-function-declaration]
  write(connfd, buff, sizeof(buff));
  ^
twoserver.c: In function 'main':
twoserver.c:37:2: warning: implicit declaration of function 'close' [-Wimplicit-function-declaration]
  close(sockfd);
  ^
root@spl13:~/205001095/ex5# ./a.out
Socket successfully created..
Socket successfully binded..
Server listening..
Server accept the client...
From server : hllllll
To client : byeeee
From server : exit
To client : exit
Server Exit...
root@spl13:~/205001095/ex5#
```


Assignment-5: Multi User Chat

Objective:

To write a program to make a chat application with multiple clients and one server to exchange messages.

Algorithm:

Server:

1. Declare an array of clients to keep track of communicating clients
2. Declare a pthread and a mutex lock for receiving messages from multiple clients
3. Create a socket address of type sockaddr_in for server and define its family as AF_INET, port number and IP address
4. Create a new file descriptor for the socket using the socket() system call. Return error if socket creation fails
5. Bind the server socket address to this socket using the bind() system call. Return error if binding fails
6. Listen to any incoming connection request using the listen() system call. Return error if listening fails
7. Accept any incoming client connection request using the accept() system call and store the return value in a client socket descriptor. Return error if accept fails.
8. Store the client socket descriptor in the array of clients
9. Create a worker thread using pthread_create and call the function recvmmsg() to receive messages from a client.
 - 9.1. Receive the message from the client using the recv() system call and save it in an array.
 - 9.2. Call the sendtoall() function using the saved message
10. In the sendtoall() function

- 10.1. Lock the mutex lock
- 10.2. Loop through the values in the clients array and find the socket descriptor of the current client.
- 10.3. Send the message to the client using send() system call. Return error if sending failure.
- 10.4. Unlock the mutex lock

Client:

- 1. Declare a pthread for receiving a message from the server
- 2. Create a socket address of type sockaddr_in for client and define its family as AF_INET, port number and IP address. Server and client must share the port number
- 3. Create a new file descriptor for the socket used for communication between server and the client using socket() system call. Return error if socket creation fails
- 4. Connect to the server socket address using the connect() system call. Return error if connection fails
- 5. Create a worker thread using pthread_create and call function recvmmsg() to receive message from the server
 - 5.1. Receive a message from the server using recv() system call.
 - 5.2. Display the received message
- 6. Send a reply to the server using the write system call.
- 7. Close the socket file descriptor

Code:

client.c:

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

int sockfd = 0;
char id[2];
pthread_t tid1,tid2;
void *read_msg(void *arg)
{
    char str[100];
    read(sockfd,str, 100);
    if(strcmp(str,"exit")==0)
    {
        printf("OK Bye!\n");
        close(sockfd);
        exit(0);
    }
    if(str[99]=='S')
    printf("%c says: %s\n",str[99],str);
    pthread_create(&tid1,NULL,read_msg,NULL);
    return (NULL);
}
void *write_msg(void *arg)
{
    char str[100];
    fflush(stdout);
    scanf("%[^\\n]%*c", str);
    str[99] = id[0];
    write(sockfd, str, 100);
    if(strcmp(str,"exit")==0)
    {
        close(sockfd);
```

```

        exit(0);
    }
    pthread_create(&tid2,NULL,write_msg,NULL);
    return (NULL);
}
int main(int argc, char *argv[])
{
    int n = 0;
    char recvBuff[1024];int temp;
    struct sockaddr_in serv_addr;
    char s[100];
    if(argc != 2)
    {
        printf("\n Usage: %s <ip of server> \n",argv[0]);
        return 1;
    }
    memset(recvBuff, '0',sizeof(recvBuff));
    if((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0){
        printf("\n Error : Could not create socket \n");
        return 1;
    }
    memset(&serv_addr, '0', sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(5000);
    printf("Server address used is: %s\n", argv[1]);
    if(inet_pton(AF_INET, argv[1], &serv_addr.sin_addr)<=0)
    {
        printf("\n inet_pton error occured\n");
        return 1;
    }
    if( connect(sockfd, (struct sockaddr *)&serv_addr,
    sizeof(serv_addr)) < 0)
    {
        printf("\n Error : Connect Failed \n");
        return 1;
    }
    read(sockfd, id, 5);
    printf("My ID : %c\n", id[0]);
    pthread_create(&tid1,NULL,read_msg,NULL);
    pthread_create(&tid2,NULL,write_msg,NULL);
    while(1);
    return 0;
}

```

server.c:

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <time.h>
#include <pthread.h>
#define MAX 5

int connfd[MAX] = {0};
int i, e[MAX] = {0};
pthread_t tid2;
pthread_t tid[MAX];
void *read_msg(void *arg)
{
    int k = *(int *)arg;
    int l;
    char str[100];
    read(connfd[k], str, 100);
    if(strcmp(str, "exit")==0)
    {
        printf("Client %c is Offline\n", (char)(65+k));
        close(connfd[k]);
        return (NULL);
    }
    for(l=0; l<i; l++)
    if(l!=k) write(connfd[l], str, 100);
    printf("%c says: %s\n", str[99], str);
    pthread_create(&tid[k], NULL, read_msg, (void *)&k);
    return (NULL);
}
void *write_msg(void *arg)
{
    int l;
    char str[100];
    fflush(stdin);
    scanf("%[^\\n]%*c", str);
    str[99] = 'S';
```



```

        for(l=0;l<i;l++)
        write(connfd[l], str, 100);
        if(strcmp(str,"exit")==0)
        exit(0);
        pthread_create(&tid2,NULL,write_msg,NULL);
        return (NULL);
    }
int main(int argc, char *argv[])
{
    int listenfd = 0;
    struct sockaddr_in serv_addr;
    char sendBuff[1025];
    int temp;
    time_t ticks;
    char C_name[MAX][2];
    int j[MAX];
    listenfd = socket(AF_INET, SOCK_STREAM, 0);
    memset(&serv_addr, '0', sizeof(serv_addr));
    memset(sendBuff, '0', sizeof(sendBuff));
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    serv_addr.sin_port = htons(5000);
    bind(listenfd, (struct sockaddr*)&serv_addr, sizeof(serv_addr));
    listen(listenfd, 10);
    for (i = 0; i < MAX; ++i){
        C_name[i][0] = (char)(65+i);
        C_name[i][1] = '\n';
        j[i] = i;
    }
    printf("My ID : S\n");
    pthread_create(&tid2,NULL,write_msg,NULL);
    for(i = 0; i < MAX; i++)
    {
        connfd[i] = accept(listenfd, (struct sockaddr*)NULL, NULL);
        write(connfd[i], C_name[i], 5);
        printf("Client %c is Online\n",C_name[i][0]);
        pthread_create(&tid[i],NULL,read_msg,(void *)(j+i));
    }
}

```

Sample I/O:

```
root@spl15: ~/Downloads/Networks/Ex5
root@spl15:~/Downloads/Networks/Ex5# gcc newclient.c -pthread
root@spl15:~/Downloads/Networks/Ex5# ./a.out 10.6.10.15
Server address used is: 10.6.10.15
My ID : C
S says: hi A
S says: Hi b
I am C
S says: Hi C
S says: Bye guys
OK Bye!
root@spl15:~/Downloads/Networks/Ex5#
```

```
root@spl15: ~/Downloads/Networks/Ex5
root@spl15:~/Downloads/Networks/Ex5# gcc newserver.c -pthread
root@spl15:~/Downloads/Networks/Ex5# ./a.out
My ID : S
Client A is Online
Client B is Online
Client C is Online
A says: hi i am A
hi A
B says: I'm B
Hi b
C says: I am C
Hi C
Bye guys
exit
```

Assignment-6: File Transfer Protocol

Objective:

Write a program to simulate the File Transfer Protocol and it should be able to share files between a server and a client.

Algorithm:

Server:

1. Declare an array of clients to keep track of communicating clients
2. Declare a pthread and a mutex lock for receiving messages from multiple clients
3. Create a socket address of type `sockaddr_in` for server and define its family as `AF_INET`, port number and IP address
4. Create a new file descriptor for the socket using the `socket()` system call. Return error if socket creation fails
5. Bind the server socket address to this socket using the `bind()` system call. Return error if binding fails
6. Listen to any incoming connection request using the `listen()` system call. Return error if listening fails
7. Accept any incoming client connection request using the `accept()` system call and store the return value in a client socket descriptor. Return error if accept fails.
8. Store the client socket descriptor in the array of clients
9. Create a worker thread using `pthread_create` and call the function `recvmsg()` to receive messages from a client.
 - 9.1. Receive the filename from the client using the `recv()` system call.
 - 9.2. Open the file in read mode using the `open()` system call
 - 9.3. If the file is not found, call the `sendtoall()` function with an error message

9.4. Else read the file and store its contents in an array and call the `sendtoall()` function with this array

10. In the `sendto all` function

10.1. Lock the mutex lock

10.2. Send the message (error or filecontents) to the client using `send()` system call. Return error if sending failure.

10.3. Unlock the mutex lock

Client:

1. Declare a pthread for receiving a message from the server

2. Create a socket address of type `sockaddr_in` for client and define its family as `AF_INET`, port number and IP address. Server and client must share the port number

3. Create a new file descriptor for the socket used for communication between server and the client using `socket()` system call. Return error if socket creation fails

4. Connect to the server socket address using the `connect()` system call. Return error if connection fails

5. Send the name of the file to the server using the `write` system call

6. Create a worker thread using `pthread_create` and call function to `recvmsg()` to receive messages from the server

6.1. Receive the file contents from the server using `recv()` system call.

6.2. Display the received data

7. Close the socket file descriptor

Code:

Server.c:

```
#include <netinet/in.h> //structure for storing address information
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h> //for socket APIs
#include <sys/types.h>
#include <string.h>
#include <unistd.h>
```

```
int main(int argc, char const* argv[])
{
    // create server socket similar to what was done in
    // client program
    int servSockD = socket(AF_INET, SOCK_STREAM, 0);

    // define server address
    struct sockaddr_in servAddr;

    servAddr.sin_family = AF_INET;
    servAddr.sin_port = htons(9081);
    servAddr.sin_addr.s_addr = INADDR_ANY;

    // bind socket to the specified IP and port
    bind(servSockD, (struct sockaddr*)&servAddr,
    sizeof(servAddr));

    // listen for connections
    listen(servSockD, 1);

    // integer to hold client socket.
    int clientSocket = accept(servSockD, NULL, NULL);

    // send's messages to client socket
    char file[255];
    recv(clientSocket, file, sizeof(file), 0);
    char cmd[100] = "ls ";
    char filecontents[1000];
    strcat(cmd, file);
    if(system(cmd)==0){
        printf("File Exists!!!");
    }
```



```

    FILE *f = fopen(file,"r");
    char ch;
    int x=0;
    do{
        ch = fgetc(f);
        filecontents[x++] = ch;
    }while(ch!=EOF);
    filecontents[x] = '\0';
} else printf("File not found");
    send(clientSocket, filecontents, strlen(filecontents), 0);
    return 0;
}

```

Client.c:

```

#include <netinet/in.h> //structure for storing address information
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h> //for socket APIs
#include <sys/types.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>

int main(int argc, char const* argv[])
{
    int sockD = socket(AF_INET, SOCK_STREAM, 0);

    struct sockaddr_in servAddr;

    servAddr.sin_family = AF_INET;
    servAddr.sin_port
    = htons(9081); // use some unused port number
    servAddr.sin_addr.s_addr = INADDR_ANY;

    int connectStatus
    = connect(sockD, (struct sockaddr*)&servAddr,
              sizeof(servAddr));

    if (connectStatus == -1) {
        printf("Error...\n");
    }
}

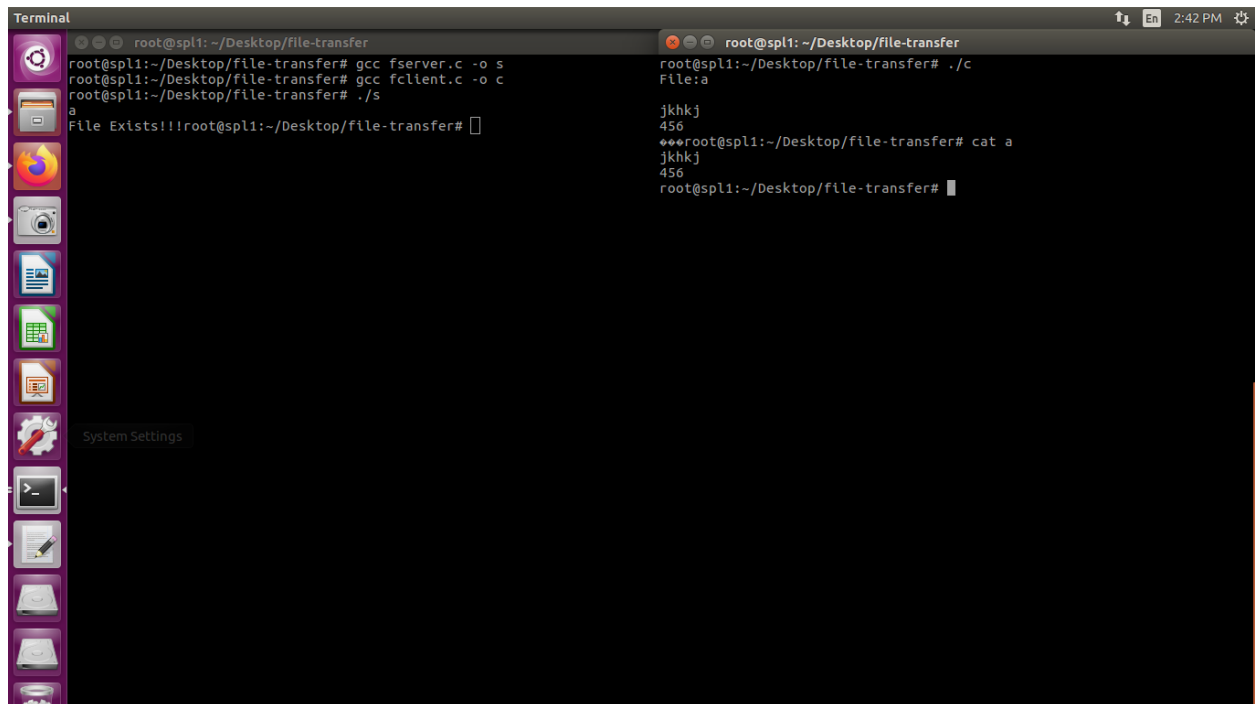
```

```

    else {
        char strData[255];
        printf("File:");
        scanf("%s",strData);
        send(sockD,strData,sizeof(strData),0);
        recv(sockD, strData, sizeof(strData), 0);
        printf("\n%s",strData);
        int nf = open("newfile.txt",O_CREAT|O_WRONLY);
        write(nf,strData,strlen(strData));
    }
    close(sockD);
    return 0;
}

```

Sample I/O:



```

Terminal
root@spl1: ~/Desktop/file-transfer
root@spl1:~/Desktop/file-transfer# gcc fserver.c -o s
root@spl1:~/Desktop/file-transfer# gcc fclient.c -o c
root@spl1:~/Desktop/file-transfer# ./s
a
File Exists!!!root@spl1:~/Desktop/file-transfer#

root@spl1: ~/Desktop/file-transfer
root@spl1:~/Desktop/file-transfer# ./c
File:a
jkhkj
456
***root@spl1:~/Desktop/file-transfer# cat a
jkhkj
456
root@spl1:~/Desktop/file-transfer#

```

Assignment-7: Simulation of ARP

Objective:

Write a program to simulate the ARP Protocol in C.

Algorithm:

Server:

1. Declare an array of clients to keep track of communicating clients
2. Declare a pthread and a mutex lock for receiving messages from multiple clients
3. Create a socket address of type sockaddr_in for server and define its family as AF_INET, port number and IP address
4. Create a new file descriptor for the socket (used for communication between server and clients) using socket() system call. Return error if socket creation fails
5. Bind the server socket address to this socket using bind() system call. Return error if binding fails
6. Listen to any incoming connection request using listen() system call. Return error if listening fails
7. Create a worker thread using pthread_create and call function to send message to all clients
 - 7.1 Read IP address from the user
 - 7.2 Lock the mutex lock
 - 7.3 Send the IP address (msg) to all connected clients (whose socket descriptors are stored in the clients array) using send() system call. Print error if sending fails
 - 7.4 Unlock the mutex lock
 - 7.5 Wait for few seconds using sleep and go to step 7.1 This repeats until user exits terminal

8. Accept the first pending request from a client using `accept()` system call and get the client's socket file descriptor. Print error if there are no pending requests
9. Lock the mutex lock
10. Add the client's socket descriptor to the array of communicating clients
11. Create a worker thread using `pthread_create` and call function to receive a message from the newly connected client
 - 11.1 Receive a message from the client using `recv()` system call. The received message is the MAC Address of the client whose IP address is sent by the server and print the same
 - 11.2 Print error if receiving message fails
12. Unlock the mutex lock
13. Close the socket file descriptor

Client:

1. Declare a pthread for receiving a message from the server
2. Create a socket address of type `sockaddr_in` for client and define its family as `AF_INET`, port number and IP address. Server and client must share the port number
3. Create a new file descriptor for the socket used for communication between server and the client using `socket()` system call. Return error if socket creation fails
4. Connect to the server socket address using `connect()` system call. Return error if connection fails
5. Create a worker thread using `pthread_create` and call function to receive message from the server
 - 5.1 Receive a message from the server using `recv()` system call. The received message is an IP address
 - 5.2 Get the IP address of the client

- 5.2.1 Create a socket to define network interface IPv4 and define its address type as AF_INET and port name using string "enp3s0"
- 5.2.2 Call the ioctl function to access the network interface information by passing the socket file descriptor and address.
- 5.2.3 Use the sin_addr attribute in ifr_addr to extract the IP address
- 5.3 Check if the message received from the server matches with the client's IP address
- 5.4 If it matches go to step 5.5. Else display that its not the client's IP address and go to step 6
- 5.5 Get the MAC address of the client
 - 5.5.1 Using getifaddrs() system call create a linked list of structures describing the network interfaces of the system
 - 5.5.2 Iterate through the linked list until the structure has an ifa_addr attribute and has sa_family as AF_PACKET
 - 5.5.3 Get the sll_addr attribute of ifa_addr and its first 6 items are stored as 6 numbers separated by ':' . This is the MAC address of the system
- 5.6 Send the MAC address (msg) to the server using send() system call. Print error if sending fails
- 5.7 Go to step 5.1 This repeats until client receives a message from the server
- 6. Close the socket file descriptor

Code:

Client.c:

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

int main(int argc, char **argv)
{
    int cli = socket(AF_INET, SOCK_STREAM, 0);

    struct sockaddr_in caddr;

    caddr.sin_family = AF_INET;
    caddr.sin_port = htons(atoi(argv[1]));
    caddr.sin_addr.s_addr = INADDR_ANY;

    connect(cli, (struct sockaddr*)&caddr, sizeof(caddr));

    system("ifconfig | cat -n | grep 20 | cut -c21-32 >nn.txt");
    char myIP[11];
    int f1 = open("nn.txt", O_RDONLY);
    char buf[1];
    int a = 0;
    while(read(f1, buf, 1))
        myIP[a++] = buf[0];
    myIP[a] = '\0';

    printf("\nClient's IP : %s", myIP);

    char IP[11];
```

```

recv(cli,IP,sizeof(IP),0);
IP[11] = '\0';
printf("\nReceived IP : %s",IP);

int flag = 1;

for(int i=0;i<11;i++)
{
    if(myIP[i] != IP[i])
    {
        flag = 0;
        printf("\nNO");
    }
}

if(flag){
    system("ifconfig | cat -n | grep 24 | tail -1 | cut -c22-39
>MAC.txt");
    char buffer[1];
    char ch[17];
    int b = 0;
    int f2 = open("MAC.txt",O_RDONLY);
    while(read(f2,buffer,1))
        ch[b++] = buffer[0];
    ch[b] = '\0';
    send(cli,ch,sizeof(ch),0);
}
else send(cli,"Incorrect!!",11,0);
close(cli);
return 0;
}

```

Server.c:

```

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

```



```

#include <time.h>
#include <ctype.h>
#include <pthread.h>

int validate_number(char *str) {
    while (*str) {
        if(!isdigit(*str)){
            return 0;
        }
        str++;
    }
    return 1;
}

int validate_ip(char *ip) {
    int i, num, dots = 0;
    char *ptr;
    if (ip == NULL)
        return 0;
    ptr = strtok(ip, ".");
    if (ptr == NULL)
        return 0;
    while (ptr) {
        if (!validate_number(ptr))
            return 0;
        num = atoi(ptr);
        if (num >= 0 && num <= 255) {
            ptr = strtok(NULL, ".");
            if (ptr != NULL)
                dots++;
        } else
            return 0;
    }
    if (dots != 3)
        return 0;
    return 1;
}

int main(int argc, char **argv)
{
    int serv = socket(AF_INET, SOCK_STREAM, 0);

    struct sockaddr_in saddr;

```

```

saddr.sin_family = AF_INET;
saddr.sin_port = htons(atoi(argv[1]));
saddr.sin_addr.s_addr = inet_addr("127.0.0.1");

bind(serv,(struct sockaddr*)&saddr,sizeof(saddr));

if(validate_ip(argv[2])==0){
    printf("[-]INVALID IP\n");
    exit(0);
}

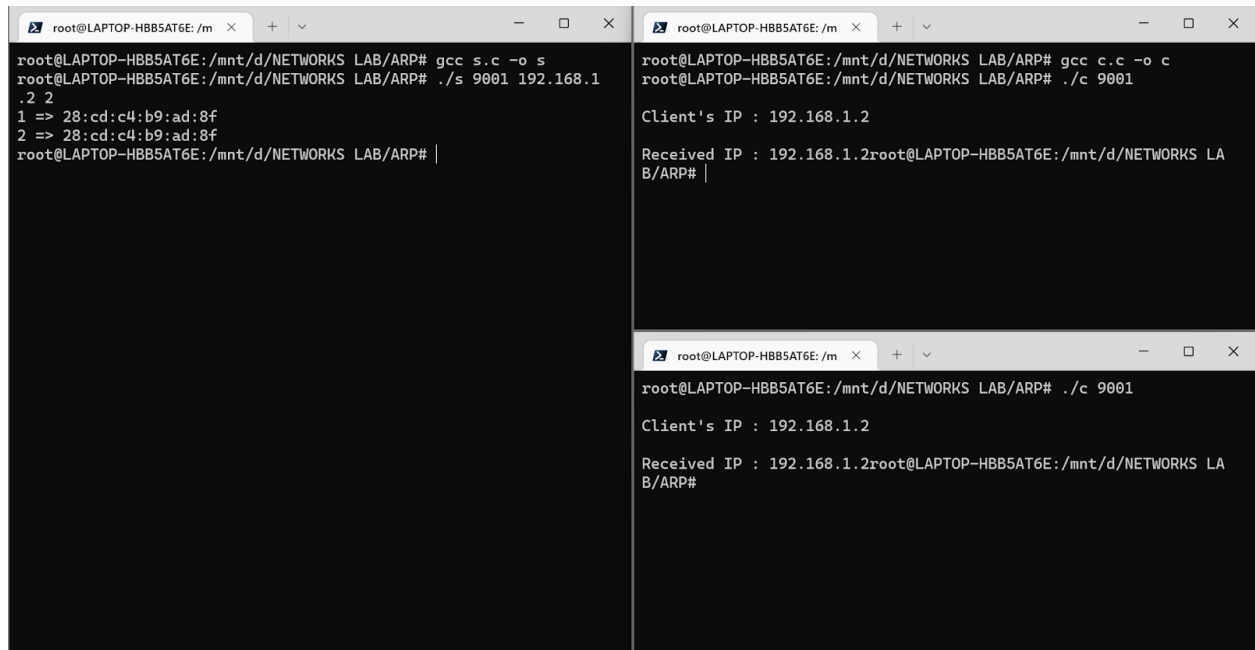
listen(serv,2);
int i = 0;
int sockets[10];

while(1){
    sockets[i++] = accept(serv,NULL,NULL);
    if(i==atoi(argv[3]))break;
}
for(int i=0;i<atoi(argv[3]);i++){

    send(sockets[i],argv[2],strlen(argv[2]),0);
    char MAC[20];
    recv(sockets[i],MAC,sizeof(MAC),0);
    printf("%d => %s\n",i+1,MAC);
}
close(serv);
return 0;
}

```

Sample I/O:



The image displays three terminal windows from a Linux system, showing the configuration and execution of a network program. The windows are arranged in a 2x2 grid, with the bottom-right cell empty.

Top-left terminal: Shows the compilation of a server program. The user is in the directory `/mnt/d/NETWORKS LAB/ARP#`. They compile `s.c` into `s` using `gcc s.c -o s`. Then, they move to `/mnt/d/NETWORKS LAB/ARP#` and run `./s 9001 192.168.1.2 2`. The output shows two MAC addresses: `1 => 28:cd:c4:b9:ad:8f` and `2 => 28:cd:c4:b9:ad:8f`.

Top-right terminal: Shows the compilation of a client program. The user is in the directory `/mnt/d/NETWORKS LAB/ARP#`. They compile `c.c` into `c` using `gcc c.c -o c`. Then, they move to `/mnt/d/NETWORKS LAB/ARP#` and run `./c 9001`. The output shows the client's IP: `Client's IP : 192.168.1.2` and the received IP: `Received IP : 192.168.1.2`.

Bottom-left terminal: Shows the execution of the client program. The user is in the directory `/mnt/d/NETWORKS LAB/ARP#`. They run `./c 9001`. The output shows the client's IP: `Client's IP : 192.168.1.2` and the received IP: `Received IP : 192.168.1.2`.

Assignment-8: Simulation of RARP

Objective:

Write a program to simulate the RARP Protocol.

Algorithm:

Server:

1. Declare an array of clients to keep track of communicating clients
2. Declare a pthread and a mutex lock for receiving messages from multiple clients
3. Create a socket address of type sockaddr_in for server and define its family as AF_INET, port number and IP address
4. Create a new file descriptor for the socket (used for communication between server and clients) using socket() system call. Return error if socket creation fails
5. Bind the server socket address to this socket using bind() system call. Return error if binding fails
6. Listen to any incoming connection request using listen() system call. Return error if listening fails
7. Create a worker thread using pthread_create and call function to send message to all clients
 - 7.1 Read MAC address from the user
 - 7.2 Lock the mutex lock
 - 7.3 Send the MAC address (msg) to all connected clients (whose socket descriptors are stored in the clients array) using send() system call. Print error if sending fails
 - 7.4 Unlock the mutex lock
 - 7.5 Wait for few seconds using sleep and go to step 7.1 This repeats until user exits terminal

8. Accept the first pending request from a client using `accept()` system call and get the client's socket file descriptor. Print error if there are no pending requests
9. Lock the mutex lock
10. Add the client's socket descriptor to the array of communicating clients
11. Create a worker thread using `pthread_create` and call function to receive a message from the newly connected client
 - 11.1 Receive a message from the client using `recv()` system call. The received message is the IP Address of the client whose MAC address is sent by the server and print the same
 - 11.2 Print error if receiving message fails
12. Unlock the mutex lock
13. Close the socket file descriptor

Client:

1. Declare a pthread for receiving a message from the server
2. Create a socket address of type `sockaddr_in` for client and define its family as `AF_INET`, port number and IP address. Server and client must share the port number
3. Create a new file descriptor for the socket used for communication between server and the client using `socket()` system call. Return error if socket creation fails
4. Connect to the server socket address using `connect()` system call. Return error if connection fails
5. Create a worker thread using `pthread_create` and call function to receive message from the server
 - 5.1 Receive a message from the server using `recv()` system call. The received message is a MAC address
 - 5.2 Get the MAC address of the client

5.2.1 Using `getifaddrs()` system call create a linked list of structures describing the network interfaces of the system

5.2.2 Iterate through the linked list until the structure has an `ifa_addr` attribute and has `sa_family` as `AF_PACKET`

5.2.3 Get the `sll_addr` attribute of `ifa_addr` and its first 6 items are stored as 6 numbers separated by ':' . This is the MAC address of the system

5.3 Check if the message received from the server matches with the client's MAC address

5.4 If it matches go to step 5.5. Else display that its not the client's MAC address and go to step 6

5.5 Get the IP address of the client

5.5.1 Create a socket to define network interface IPv4 and define its address type as `AF_INET` and port name using string "enp3s0"

5.5.2 Call the `ioctl` function to access the network interface information by passing the socket file descriptor and address.

5.5.3 Use the `sin_addr` attribute in `ifr_addr` to extract the IP address

5.6 Send the IP address (`msg`) to the server using `send()` system call. Print error if sending fails

5.7 Go to step 5.1 This repeats until client receives a message from the server

6. Close the socket file descriptor

Code:

CLIENT:

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

int main(int argc, char **argv)
{
    int cli = socket(AF_INET, SOCK_STREAM, 0);

    struct sockaddr_in caddr;

    caddr.sin_family = AF_INET;
    caddr.sin_port = htons(atoi(argv[1]));
    caddr.sin_addr.s_addr = INADDR_ANY;

    connect(cli, (struct sockaddr*)&caddr, sizeof(caddr));

    system("ifconfig | cat -n | grep 24 | tail -1 | cut -c22-39 >MAC.txt");
    char buffer[1];
    char ch[17];
    int b = 0;
    int f2 = open("MAC.txt", O_RDONLY);
    while(read(f2, buffer, 1))
        ch[b++] = buffer[0];
    ch[b] = '\0';

    printf("\nClient's MAC: %s", ch);
}
```



```

char MAC[17];
recv(cli,MAC,sizeof(MAC),0);
MAC[sizeof(MAC)] = '\0';
printf("\nReceived MAC : %s\n",MAC);
int flag = 1;
for(int i=0;i<17;i++)
{
    if(ch[i] != MAC[i])
    {
        flag = 1;
        printf("\nNO!!");
        break;
    }
}
if(flag == 1)
{
    system("ifconfig | cat -n | grep 20 | cut -c21-32 >nn.txt");
    char myIP[11];
    int f1 = open("nn.txt",O_RDONLY);
    char buf[1];
    int a = 0;
    while(read(f1,buf,1))
        myIP[a++] = buf[0];
    myIP[a] = '\0';
    send(cli,myIP,sizeof(myIP),0);
}
else send(cli,"Mismatch!!",10,0);
return 0;
}

```

SERVER:

```

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

```

```

#include<ctype.h>
#include <pthread.h>
int MACvalidator(char MAC[17],int n)
{
    if(n != 17) return 0;
    for(int i=0;i<17;i++)
    {
        if((i+1)%3 != 0 && !isalnum(MAC[i])) return 0;
        if((i+1)%3 == 0 && MAC[i]!=':') return 0;
    }
    return 1;
}
int countchar(char str[])
{
    int c = 0;
    while(str[c]!='\0') c++;
    return c;
}

int main(int argc,char **argv)
{
    int serv = socket(AF_INET,SOCK_STREAM,0);

    struct sockaddr_in saddr;

    saddr.sin_family = AF_INET;
    saddr.sin_port = htons(atoi(argv[1]));
    saddr.sin_addr.s_addr = inet_addr("127.0.0.1");

    bind(serv,(struct sockaddr*)&saddr,sizeof(saddr));

    listen(serv,2);
    int i = 0;
    int sockets[10];
    int var = countchar(argv[2]);
    printf("\n%d",var);
    if(MACvalidator(argv[2],var) == 0)
    {
        printf("\nINVALID MAC!!");
        return 0;
    }
    printf("\nVALID MAC");
}

```

```

while(1){
    sockets[i++] = accept(serv, NULL, NULL);
    if(i==atoi(argv[3]))break;
}
printf("\n");
for(int i=0;i<atoi(argv[3]);i++){

    send(sockets[i], argv[2], strlen(argv[2]), 0);
    char MAC[11];
    recv(sockets[i], MAC, 11, 0);
    MAC[sizeof(MAC)] = '\0';
    printf("%d => %s\n", i+1, MAC);
}
close(serv);
return 0;
}

```

Sample I/O:

```

root@LAPTOP-HBB5AT6E:/mnt/d/NETWORKS LAB/RARP# gcc rs.c -o s
root@LAPTOP-HBB5AT6E:/mnt/d/NETWORKS LAB/RARP# ./s 9001 28:cd:c4:b9:ad:8f 2
17
VALID MAC
1 => 192.168.1.4
2 => 192.168.1.4
root@LAPTOP-HBB5AT6E:/mnt/d/NETWORKS LAB/RARP#

root@LAPTOP-HBB5AT6E:/mnt/d/NETWORKS LAB/RARP# gcc rc.c -o c
root@LAPTOP-HBB5AT6E:/mnt/d/NETWORKS LAB/RARP# ./c 9001

Client's MAC: 28:cd:c4:b9:ad:8f

Received MAC : 28:cd:c4:b9:ad:8f
root@LAPTOP-HBB5AT6E:/mnt/d/NETWORKS LAB/RARP#

root@LAPTOP-HBB5AT6E:/mnt/d/NETWORKS LAB/RARP# ./c 9001

Client's MAC: 28:cd:c4:b9:ad:8f

Received MAC : 28:cd:c4:b9:ad:8f
root@LAPTOP-HBB5AT6E:/mnt/d/NETWORKS LAB/RARP#

```

Assignment-9: Simulation of Error Correction using Hamming Code

Objective:

Write a program to simulate error correction using hamming code.

Algorithm:

hamming.h package with functions (for error correction):

(countbits) function to count the number of bits

1. While number is greater than 0

1.1. Divide number by 10

1.2. Increment count

(binary) function to convert number to binary

1. While number is greater than 0

1.1. Store number%2 in r

1.2. Increment bin by $r \cdot i^{10}$

1.3. Divide number by 2

1.4. Increment i

(ispresent) function to check if num is present at position pos

1. Iterating through each pos from 0

1.1. Store $\text{num} \% 10$ in rem

1.2. Divide num by 10

2. Return 1 if rem=1

(ispower2) function to check if n is a power of 2

1. Return 1 if ceil of $\log n$ to base 2 is equal to floor of $\log n$ to base 2

(decimal) function to check if num is in decimal

1. While number is greater than 0

1.1. Store $\text{number} \% 10$ in rem

1.2. Increment result by $\text{rem} * i^2$

1.3. Divide num by 10

1.4. Increment i

Server:

1. Declare an array of clients to keep track of communicating clients
2. Declare a pthread and a mutex lock for receiving messages from multiple clients
3. Create a socket address of type `sockaddr_in` for server and define its family as `AF_INET`, port number and IP address
4. Create a new file descriptor for the socket (used for communication between server and clients) using `socket()` system call. Return error if socket creation fails
5. Bind the server socket address to this socket using `bind()` system call. Return error if binding fails
6. Listen to any incoming connection request using `listen()` system call. Return error if listening fails
7. In a loop for each client
 - 7.1. Accept each client.
 - 7.2. Create a thread for each client.
 - 7.3. Call `(recvmsg)` function for each client thread.
8. Unlock the mutex lock

Function `recvmsg`:

1. For each client
 - 1.1. Accept the message and print it
 - 1.2. Convert the message to integer using `atoi()`
 - 1.3. Store number of bits of the message in total
 - 1.4. Store each digit of the message in array `arr`

- 1.5. Identify any errors in the message to be stored in variable binary
- 1.6. Check for any errors in the message using decimal function from hamming.h package
 - 1.6.1. Print error if present, in bit pointed by variable error
 - 1.6.2. Decode the error message and print decoded message in array newarr

Client:

1. Declare a pthread for receiving a message from the server
2. Create a socket address of type sockaddr_in for client and define its family as AF_INET, port number and IP address. Server and client must share the port number
3. Create a new file descriptor for the socket used for communication between server and the client using socket() system call. Return error if socket creation fails
4. Connect to the server socket address using connect() system call. Return error if connection fails
5. Create a client thread which is always waiting for a message
6. Enter data/message to be sent and convert it to long integer
7. Store number of bits in message in variable n
8. Store number of bits of the message in total
9. Store each digit of the message in array arr
10. Find number of redundant bits in the message
11. Accept bit position to introduce error if needed and print encoded message after introducing error
12. Send the encoded message to the server
13. Close the thread and socket file descriptor

Code:

CLIENT:

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

```

#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <fcntl.h>
#include <arpa/inet.h>
#include <sys/ioctl.h>
#include <net/if.h>

int main(int argc, char **argv)
{
    int cli = socket(AF_INET, SOCK_STREAM, 0);

    struct sockaddr_in caddr;

    caddr.sin_family = AF_INET;
    caddr.sin_port = htons(atoi(argv[1]));
    caddr.sin_addr.s_addr = INADDR_ANY;

    connect(cli, (struct sockaddr*)&caddr, sizeof(caddr));

    system("ifconfig | cat -n | grep 24 | tail -1 | cut -c22-39 >MAC.txt");
    char buffer[1];
    char ch[17];
    int b = 0;
    int f2 = open("MAC.txt", O_RDONLY);
    while(read(f2, buffer, 1))
        ch[b++] = buffer[0];
    ch[b] = '\0';

    printf("\nClient's MAC: %s", ch);

    char MAC[17];
    recv(cli, MAC, sizeof(MAC), 0);
    MAC[sizeof(MAC)] = '\0';
    printf("\nReceived MAC : %s\n", MAC);
    int flag = 1;
    for(int i=0; i<17; i++)
    {
        if(ch[i] != MAC[i])
        {
            flag = 1;

```

```

        printf("\nNO!!");
        break;
    }
}
if(flag == 1)
{
    system("ifconfig | cat -n | grep 20 | cut -c21-32 >nn.txt");
    char myIP[11];
    int f1 = open("nn.txt",O_RDONLY);
    char buf[1];
    int a = 0;
    while(read(f1,buf,1))
        myIP[a++] = buf[0];
    myIP[a] = '\0';
    send(cli,myIP,sizeof(myIP),0);
}
else send(cli,"Mismatch!!",10,0);
return 0;
}

```

SERVER:

```

#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <time.h>
#include <ctype.h>
#include <ctype.h>
#include <pthread.h>
int MACvalidator(char MAC[17],int n)
{
    if(n != 17) return 0;
    for(int i=0;i<17;i++)
    {
        if((i+1)%3 != 0 && !isalnum(MAC[i])) return 0;
        if((i+1)%3 == 0 && MAC[i]!=':') return 0;
    }
}

```



```

    return 1;
}
int countchar(char str[])
{
    int c = 0;
    while(str[c]!='\0') c++;
    return c;
}

int main(int argc, char **argv)
{
    int serv = socket(AF_INET, SOCK_STREAM, 0);

    struct sockaddr_in saddr;

    saddr.sin_family = AF_INET;
    saddr.sin_port = htons(atoi(argv[1]));
    saddr.sin_addr.s_addr = inet_addr("127.0.0.1");

    bind(serv, (struct sockaddr*)&saddr, sizeof(saddr));

    listen(serv, 2);
    int i = 0;
    int sockets[10];
    int var = countchar(argv[2]);
    printf("\n%d", var);
    if(MACvalidator(argv[2], var) == 0)
    {
        printf("\nINVALID MAC!!");
        return 0;
    }
    printf("\nVALID MAC");

    while(1){
        sockets[i++] = accept(serv, NULL, NULL);
        if(i==atoi(argv[3]))break;
    }
    printf("\n");
    for(int i=0; i<atoi(argv[3]); i++){

        send(sockets[i], argv[2], strlen(argv[2]), 0);
        char MAC[11];
        recv(sockets[i], MAC, 11, 0);
    }
}

```

```

        MAC[sizeof(MAC)] = '\0';
        printf("%d => %s\n",i+1,MAC);
    }
    close(serv);
    return 0;
}

```

Sample I/O:

```

root@LAPTOP-HBB5AT6E:/mnt/d/NETWORKS LAB/RARP# gcc rs.c -o s
root@LAPTOP-HBB5AT6E:/mnt/d/NETWORKS LAB/RARP# ./s 9001 28:cd:c4:b9:ad:8f 2
17
VALID MAC
1 => 192.168.1.4
2 => 192.168.1.4
root@LAPTOP-HBB5AT6E:/mnt/d/NETWORKS LAB/RARP#

root@LAPTOP-HBB5AT6E:/mnt/d/NETWORKS LAB/RARP# gcc rc.c -o c
root@LAPTOP-HBB5AT6E:/mnt/d/NETWORKS LAB/RARP# ./c 9001

Client's MAC: 28:cd:c4:b9:ad:8f

Received MAC : 28:cd:c4:b9:ad:8f
root@LAPTOP-HBB5AT6E:/mnt/d/NETWORKS LAB/RARP#

root@LAPTOP-HBB5AT6E:/mnt/d/NETWORKS LAB/RARP# ./c 9001

Client's MAC: 28:cd:c4:b9:ad:8f

Received MAC : 28:cd:c4:b9:ad:8f
root@LAPTOP-HBB5AT6E:/mnt/d/NETWORKS LAB/RARP#

```

Assignment-10: DNS

Objective:

Write a program to simulate the DNS using UDP.

Algorithm:

Server:

- 1) Create a socket file descriptor
- 2) Set socket options
- 3) Bind the socket with the server address
- 4) Start a while loop
 - a) Empty the buffer
 - b) Receive message from clients using "recvfrom" function (UDP) with flag MSG_WAITALL.
 - c) Store the message in a variable x
 - d) Traverse the dns table to check whether the x is present
 - i) if yes, return the most recent ip address mapped to the x.
 - ii) if no, prompt user to make a new entry to the dns table
 - e) Send the returned ip address to the client using sendto function with the flag "MSG_CONFIRM"
 - f) If user chooses to append a new IP address to one of the domain y in dns table
 - i) traverse the dns table to find y, and get the ip address array mapped to the y and append the new ip address to this array.
 - g) If user enters exit, go out of the while loop.
- 5) Close server socket and terminate the program.

Client:

- 1) Create a socket file descriptor with the flag "SOCK_DGRAM"
- 2) Set socket options
- 3) Accept domain name from the user.
- 4) Empty the buffer
- 5) Send the domain name to the server using "sendto" function using the flag "MSG_CONFIRM" .
- 6) Empty the buffer.
- 7) Receive the message from the server using the function "recvfrom" with the flag "MSG_WAITALL"
- 8) Print the received IP address and close the client socket.

Code:

CLIENT:

```
// Client 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

// 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;

char name[50];
printf("\nEnter domain name : ");
scanf("%s",name);

sendto(sockfd, (char *)name, strlen(name),

MSG_CONFIRM, (const struct sockaddr *) &servaddr,

        sizeof(servaddr));

printf("DN 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;
}

```

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 <netinet/in.h>
#include <ctype.h>

#define PORT      8080
#define MAXLINE 1024
int t = 2;
typedef struct domain
{
    char name[50];
    char IP[25];
    char repeated[10][25];
    int pointer;
}DNS;
// Driver code

int validate_number(char *str) {
    while (*str) {
        if(!isdigit(*str)){
            return 0;
        }
        str++;
    }
    return 1;
}

int validate_ip(char *ip) {
    int i, num, dots = 0;
    char *ptr;
    if (ip == NULL)
        return 0;
}

```

```

    ptr = strtok(ip, ".");
    if (ptr == NULL)
        return 0;
while (ptr) {
    if (!validate_number(ptr))
        return 0;
    num = atoi(ptr);
    if (num >= 0 && num <= 255) {
        ptr = strtok(NULL, ".");
        if (ptr != NULL)
            dots++;
    } else
        return 0;
    }
    if (dots != 3)
        return 0;
    return 1;
}

int main() {

    int sockfd;

    char buffer[MAXLINE];

    char *hello = "Hello from server";

    struct sockaddr_in servaddr, cliaddr;

    // 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));

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


// Filling server information

servaddr.sin_family      = AF_INET; // IPv4

servaddr.sin_addr.s_addr = INADDR_ANY;

servaddr.sin_port = htons(PORT);


// Bind the socket with the server address

if ( bind(sockfd, (const struct sockaddr *)&servaddr,
        sizeof(servaddr)) < 0 )
{
    perror("bind failed");
    exit(EXIT_FAILURE);
}


int len, n;

DNS node[10];
strcpy(node[0].name, "www.google.com");
strcpy(node[0].IP, "192.0.0.1");
node[0].pointer = 0;
strcpy(node[0].repeated[node[0].pointer++], "192.0.0.1");


strcpy(node[1].name, "www.yahoo.com");

```



```

strcpy(node[1].IP, "192.0.3.2");
node[1].pointer = 0;
strcpy(node[1].repeated[node[1].pointer++], "192.0.3.2");

len = sizeof(cliaddr); //len is value/result

while(1){

printf("\ntable : \n");
//printf("Hello message sent.\n");
for(int i=0;i<t;i++)
{
printf("\n%s\t%s\n",node[i].name,node[i].IP);
for(int j = 0;j<node[i].pointer;j++)
{
printf("%s ",node[i].repeated[j]);
}
printf("\n");
}

n = recvfrom(sockfd, (char *)buffer, MAXLINE,

MSG_WAITALL, ( struct sockaddr *) &cliaddr,

&len);
int ch;
printf("\nEnter choice:\nCheck(0) \nModify(1)\n");
printf("\nEnter choice : ");
scanf("%d",&ch);

buffer[n] = '\0';
if(ch == 0){

char res[50];
int flag = 0;
for(int i=0;i<t;i++)
{
if(strcmp(node[i].name,buffer) == 0)
{
flag = 1;
printf("\n%s",node[i].IP);

```

```

        strcpy(res,node[i].IP);
        break;
    }
}
if(flag == 0) strcpy(res,"Not Found");

printf("Client : %s\n", buffer);
//printf("\n%s",res);

sendto(sockfd, (char *)res, strlen(res),

MSG_CONFIRM, (const struct sockaddr *) &cliaddr,

len);
}
else
{
    printf("\nEnter IP for %s : ",buffer);
    char ip[25];
    scanf("%s",ip);
    char temp[255];
    strcpy(temp,ip);
    if(validate_ip(ip) == 0)
        printf("\nInvalid IP");
    else
    {
        int flag = 0;
        for(int i=0;i<t;i++)
        {
            if(strcmp(node[i].name,buffer) == 0)
            {
                flag = 1;
                //printf("\n%s",node[i].IP);
                strcpy(node[i].repeated[node[i].pointer++],temp);
                strcpy(node[i].IP,node[i].repeated[node[i].pointer-1]);
                break;
            }
        }
        if(flag == 0)
        {
            node[t].pointer = 0;
            strcpy(node[t].name,buffer);

```

```

        strcpy(node[t].IP,temp);
        strcpy(node[t].repeated[node[t].pointer++],temp);
        t++;
    }
}
sendto(sockfd, "Updated", strlen("Updated"),

MSG_CONFIRM, (const struct sockaddr *) &cliaddr,

len);
}

}

return 0;
}

```

SAMPLE I/O:

```

root@spl2:~/Desktop/R_DNS# ./S
table :

www.google.com  192.0.0.1
192.0.0.1

www.yahoo.com   192.0.3.2
192.0.3.2

Enter choice:
Check(0)
Modify(1)

Ener choice : 0

192.0.0.1Client : www.google.com

table :

www.google.com  192.0.0.1
192.0.0.1

www.yahoo.com   192.0.3.2

```

```

table :

www.google.com  192.0.0.1
192.0.0.1

www.yahoo.com   192.0.3.2
192.0.3.2

Enter choice:
Check(0)
Modify(1)

Ener choice : 1

Enter IP for www.bing.com : 1.2.3.4

table :

www.google.com  192.0.0.1
192.0.0.1

www.yahoo.com   192.0.3.2
192.0.3.2

```

CLIENT:

```

root@spl2:~/Desktop/R_DNS# ./c

Enter domain name : www.google.com
DN sent.
Server : 192.0.0.1
root@spl2:~/Desktop/R_DNS# ./c

Enter domain name : www.bing.com
DN sent.
Server : Updated
root@spl2:~/Desktop/R_DNS# █

```

Assignment-11: Webpage Download

Objective:

To implement HTTP web client program to download a file using socket programming

Algorithm:

1. Read the name of the server as command line argument
 - i. Loop through the name of the server to find the first '/' char
 - ii. Split the name of the server to obtain hostname and trail separately
2. Get the address of the server using gethostbyname() that returns the pointer to network data structure for given host.
 - i. Store the value returned by gethostbyname() in the variable of type struct hostent
 - ii. If null is returned throw an error and terminate program
3. Store the ip of the first network address in host in the ip buffer.
4. Create a TCP socket using socket()
 - i. If -1 is returned throw error and terminate program
5. Connect to remote server using connect()
 - i. If returned value is lesser than 0, throw error and terminate program
6. Send request using a GET /path/filename HTTP/1.1\r\n request using either send() or write().
7. Receive the response using either recv() or read().
8. Parse the response to find out if the request succeeded and what format the file data is being sent as.
9. Write the downloaded page into file under a different name in a local folder.
10. Close the socket and the file.

Code:

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

```

int main(int argc, char *argv[])
{
    int socket_desc, i, bytes_read;
    char server_reply[1024], ip[100],
request[100],trail[100],filename[50];
    char *hostname = argv[1];
    struct sockaddr_in server;
    struct hostent *he;
    struct in_addr **addr_list;
    FILE *fp;
    /*Splitting the trailer part*/
    strcpy(trail,"\0");

    int l = 0;

    for(l=0;l<strlen(hostname);l++)
        if(hostname[l] == '/')
        {
            break;
        }

    if(l!=strlen(hostname))
    {
        strcpy(trail,hostname+l);
        hostname[l] = '\0';
        printf("Hostname = %s, file =
%s",hostname,trail); }

    if ((he = gethostbyname(hostname)) == NULL)
    { //gethostbyname failed
        perror("gethostbyname\n");
        return 1;
    }

    addr_list = (struct in_addr **) he->h_addr_list;

    for(i = 0; addr_list[i] != NULL; i++) {
        //Return the first one;
        strcpy(ip , inet_ntoa(*addr_list[i]) );
    }

```

```

//Create socket
socket_desc = socket(AF_INET, SOCK_STREAM, 0); if
(socket_desc == -1) {
printf("Could not create socket!\n");
}

server.sin_addr.s_addr = inet_addr(ip);
server.sin_family = AF_INET;
server.sin_port = htons(80);

//Connect to remote server
if (connect(socket_desc , (struct sockaddr
*)&server , sizeof(server)) < 0) {
printf("connect error!\n");
return 1;
}

printf("Connected...\n");

for(int i=0;hostname[i]!='\0';i++)
//Send some data
snprintf(request, 99, "GET %s
HTTP/1.1\r\n" "Host: %s\r\n"
"\r\n\r\n", trail,hostname
);
if (send(socket_desc, request, strlen(request), 0)
< 0) { puts("Send failed!\n");
return 1;
}
puts("Data Sent...\n");

//Receive a reply from the server
printf("\nEnter the filename: ");
scanf(" %s",filename);

fp = fopen(filename, "w+");

while (bytes_read = read(socket_desc,
server_reply, sizeof(server_reply)) > 0) {

```

```

        fputs(server_reply, fp);
        memset(server_reply, 0,
        sizeof(server_reply)); }
        do {
            bytes_read = read(socket_desc,
server_reply, sizeof(server_reply));
            fputs(server_reply, fp);
            memset(server_reply, 0,
            sizeof(server_reply)); } while
            (bytes_read > 0);

        printf("reply received...\n");

        fclose(fp);
        close(socket_desc);
        return 0;
    }

```

Sample I/O:

HTTP/1.1 301 Moved Permanently Date: Wed, 02 Nov 2022 09:19:29 GMT Server: Apache Location: <https://www.ssn.edu.in/wp-content/uploads/2021/02/NIRF-2021-SSNCE-ENG.pdf> Content-Length: 282 Connection: close Content-Type: text/html; charset=iso-8859-1

Moved Permanently

The document has moved [here](#).

National Institutional Ranking Framework
Ministry of Education
Government of India
Welcome to Data Capturing System: ENGINEERING

Submitted Institute Data for NIRF2021
Institute Name: Sri Sivasubramaniya Nadar College of Engineering [IR-E-C-16604]

Sanctioned (Approved) Intake

Academic Year	2019-20	2018-19	2017-18	2016-17	2015-16	2014-15
UG [4 Years Program(s)]	780	960	900	780	-	-
PG [2 Year Program(s)]	180	216	-	-	-	-

Total Actual Student Strength (Programs) Offered by Your Institution

(All programs of all years)	No. of Male Students	No. of Female Students	Total Students	Within State (including male & female)	Outside State (including male & female)	Outside Country (including male & female)	Economically backward (including male & female)	Socially Challenged (SC+ST+OBC including male & female)	No. of students receiving full tuition fee reimbursement from the State and Central Government	No. of students receiving full tuition fee reimbursement from Institution Funds	No. of students receiving full tuition fee reimbursement from the Private Bodies	No. of students who are not receiving full tuition fee reimbursement
UG [4 Years Program(s)]	2524	1440	3964	3437	469	58	725	1205	371	569	203	787
PG [2 Year Program(s)]	48	90	138	129	8	1	60	40	20	5	5	90

Placement & Higher Studies

UG [4 Years Program(s)]: Placement & higher studies for previous 3 years

Academic Year	No. of first year students intake in the year	No. of first year students admitted in the year	Academic Year	No. of students admitted through Lateral entry	Academic Year	No. of students graduating in minimum stipulated time	No. of students placed	Median salary of placed graduates(Amount in Rs.)	No. of students selected for Higher Studies
2014-15	619	829	2015-16	101	2017-18	800	633	4090000(four lakhs)	120
2015-16	619	826	2016-17	101	2018-19	787	642	4480000(four lakhs forty thousand)	123
2016-17	780	830	2017-18	101	2019-20	703	579	5090000(Five lakhs)	107

PG [2 Years Program(s)]: Placement & higher studies for previous 3 years

Academic Year	No. of first year students intake in the year	No. of first year students admitted in the year	Academic Year	No. of students admitted through Lateral entry	Academic Year	No. of students graduating in minimum stipulated time	No. of students placed	Median salary of placed graduates(Amount in Rs.)	No. of students selected for Higher Studies
---------------	---	---	---------------	--	---------------	---	------------------------	--	---

Assignment-12: Simulation of Simple Topology using TCP/UDP in ns2

Objective:

Write a tcl program to create a simple networks and simulate flows using tcp and udp in ns2.

Code:

```
set ns [new Simulator]
$ns rtproto LS

set node1 [$ns node]
set node2 [$ns node]
set node3 [$ns node]
set node4 [$ns node]
set node5 [$ns node]
set node6 [$ns node]

set tf [open out.tr w]
$ns trace-all $tf
set nf [open out.nam w]
$ns namtrace-all $nf

$node1 label "node 1"
$node2 label "node 2"
$node3 label "node 3"
$node4 label "node 4"
$node5 label "node 5"
$node6 label "node 6"

$ns color 1 blue
$ns color 2 red

$ns duplex-link $node1 $node3 1.5Mb 10ms DropTail
$ns duplex-link $node2 $node3 1.5Mb 10ms DropTail
$ns duplex-link $node3 $node4 0.5Mb 10ms DropTail
$ns duplex-link $node4 $node5 1.5Mb 10ms DropTail
$ns duplex-link $node4 $node6 1.5Mb 10ms DropTail

$ns queue-limit $node3 $node4 10

set tcp [new Agent/TCP]
$ns attach-agent $node1 $tcp
```

```

set sink [new Agent/TCPSink]
$ns attach-agent $node4 $sink
$ns connect $tcp $sink

set udp [new Agent/UDP]
$ns attach-agent $node1 $udp
set null [new Agent/Null]
$ns attach-agent $node5 $null
$ns connect $udp $null
$udp set fid_ 2

$ns attach-agent $node4 $sink
$ns connect $tcp $sink

set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set packetsize_ 100
$cbr set rate_ 0.01Mb
$cbr set random_ false

$tcp set packetsize_ 552

$tcp set fid_ 1

$tcp set color 1
$udp set color 2

set traffic_ftp2 [new Application/FTP]
$traffic_ftp2 attach-agent $tcp

proc finish {} {

global ns nf
$ns flush-trace
close $nf
exec nam out.nam &
exit 0

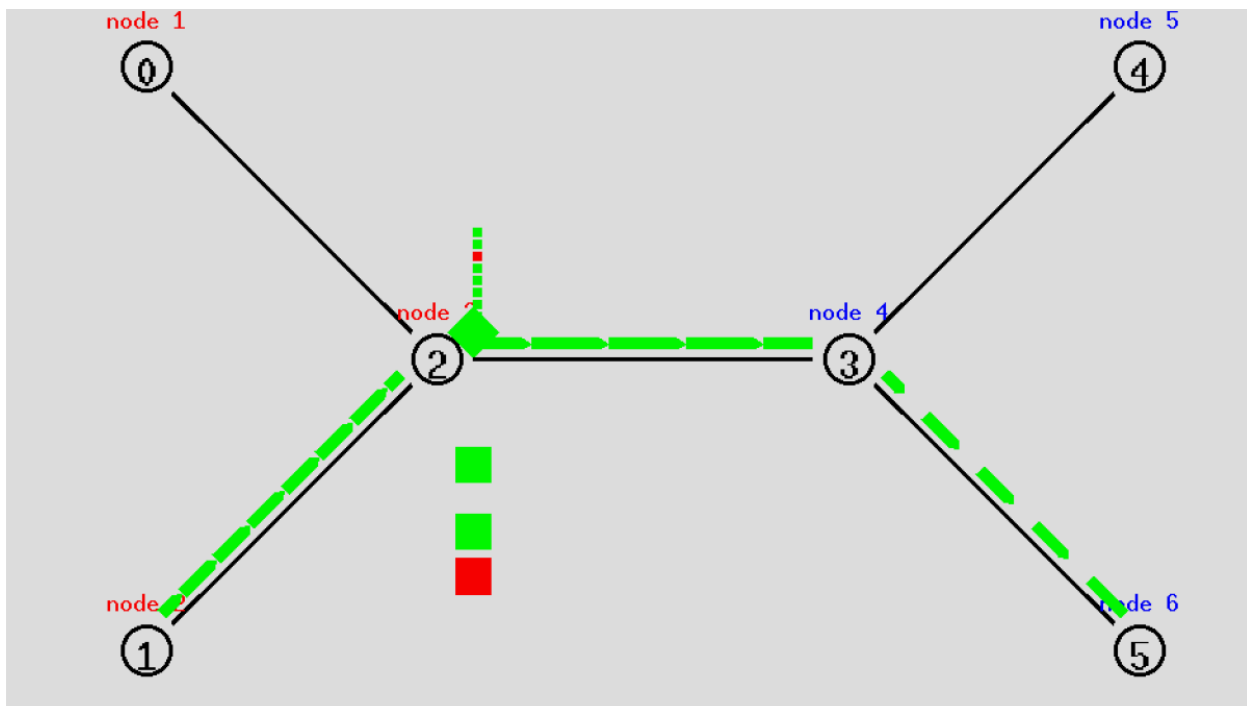
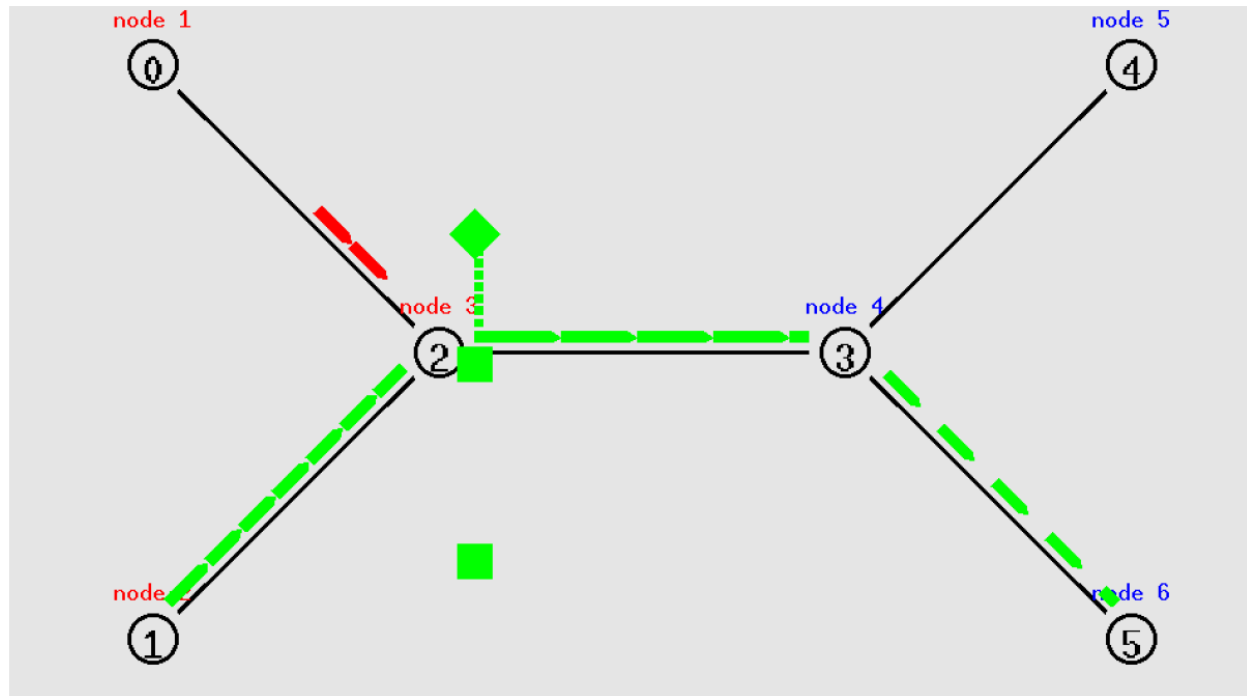
}

$ns at 0.5 "$traffic_ftp2 start"
$ns rtmodel-at 1.0 down $node2 $node3
$ns rtmodel-at 2.0 up $node2 $node3

```

```
$ns at 3.0 "$traffic_ftp2 start"
$ns at 4.0 "$traffic_ftp2 stop"
$ns at 5.0 "finish"
$ns run
```

Sample I/O:



Assignment-13: Routing Protocols

Objective:

To Simulate different Routing protocols like link state and distance vector in ns2.

Code(LS):

```
set ns [new Simulator]
$ns rtproto LS
```

```
set node1 [$ns node]
set node2 [$ns node]
set node3 [$ns node]
set node4 [$ns node]
set node5 [$ns node]
set node6 [$ns node]
```

```
set tf [open out.tr w]
$ns trace-all $tf
set nf [open out.nam w]
$ns namtrace-all $nf
```

```
$node1 label "node 1"
$node2 label "node 2"
$node3 label "node 3"
$node4 label "node 4"
$node5 label "node 5"
$node6 label "node 6"
```

```
$ns color 1 blue
$ns color 2 red
```

```
$ns duplex-link $node1 $node3 1.5Mb 10ms DropTail
$ns duplex-link $node2 $node3 1.5Mb 10ms DropTail
$ns duplex-link $node3 $node4 0.5Mb 10ms DropTail
$ns duplex-link $node4 $node5 1.5Mb 10ms DropTail
$ns duplex-link $node4 $node6 1.5Mb 10ms DropTail
```

```
$ns queue-limit $node3 $node4 10
```

```
set tcp [new Agent/TCP]
```

```

$ns attach-agent $node1 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $node4 $sink
$ns connect $tcp $sink

set udp [new Agent/UDP]
$ns attach-agent $node1 $udp
set null [new Agent/Null]
$ns attach-agent $node5 $null
$ns connect $udp $null
$udp set fid_ 2

$ns attach-agent $node4 $sink
$ns connect $tcp $sink

set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set packetsize_ 100
$cbr set rate_ 0.01Mb
$cbr set random_ false

$tcp set packetsize_ 552

$tcp set fid_ 1

$tcp set color 1
$udp set color 2

set traffic_ftp2 [new Application/FTP]
$traffic_ftp2 attach-agent $tcp

proc finish {} {

global ns nf
$ns flush-trace
close $nf
exec nam out.nam &
exit 0

}

$ns at 0.5 "$traffic_ftp2 start"
$ns rtmodel-at 1.0 down $node2 $node3

```

```
$ns rtmodel-at 2.0 up $node2 $node3
$ns at 3.0 "$traffic_ftp2 start"
$ns at 4.0 "$traffic_ftp2 stop"
$ns at 5.0 "finish"
$ns run
```

DV:

```
set ns [new Simulator]
$ns rtproto DV
```

```
set node1 [$ns node]
set node2 [$ns node]
set node3 [$ns node]
set node4 [$ns node]
set node5 [$ns node]
set node6 [$ns node]
```

```
set tf [open out.tr w]
$ns trace-all $tf
set nf [open out.nam w]
$ns namtrace-all $nf
```

```
$node1 label "node 1"
$node2 label "node 2"
$node3 label "node 3"
$node4 label "node 4"
$node5 label "node 5"
$node6 label "node 6"
```

```
$ns color 1 blue
$ns color 2 red
```

```
$ns duplex-link $node1 $node3 1.5Mb 10ms DropTail
$ns duplex-link $node2 $node3 1.5Mb 10ms DropTail
$ns duplex-link $node3 $node4 0.5Mb 10ms DropTail
$ns duplex-link $node4 $node5 1.5Mb 10ms DropTail
$ns duplex-link $node4 $node6 1.5Mb 10ms DropTail
```

```
$ns queue-limit $node3 $node4 10
```

```

set tcp [new Agent/TCP]
$ns attach-agent $node1 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $node4 $sink
$ns connect $tcp $sink

set udp [new Agent/UDP]
$ns attach-agent $node1 $udp
set null [new Agent/Null]
$ns attach-agent $node5 $null
$ns connect $udp $null
$udp set fid_ 2

$ns attach-agent $node4 $sink
$ns connect $tcp $sink

set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set packetsize_ 100
$cbr set rate_ 0.01Mb
$cbr set random_ false

$tcp set packetsize_ 552

$tcp set fid_ 1

$tcp set color 1
$udp set color 2

set traffic_ftp2 [new Application/FTP]
$traffic_ftp2 attach-agent $tcp

proc finish {} {

global ns nf
$ns flush-trace
close $nf
exec nam out.nam &
exit 0

}

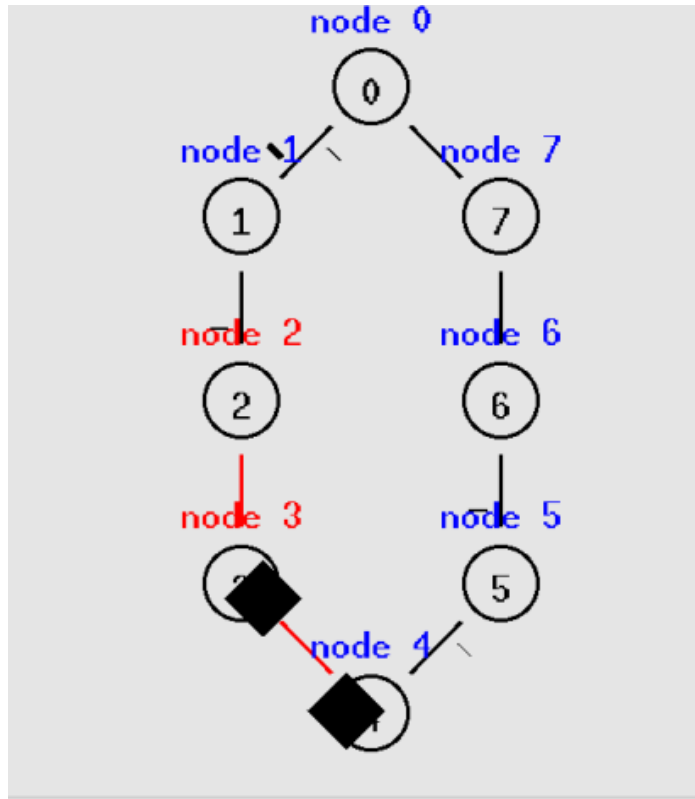
$ns at 0.5 "$traffic_ftp2 start"

```



```
$ns rtmodel-at 1.0 down $node2 $node3
$ns rtmodel-at 2.0 up $node2 $node3
$ns at 3.0 "$traffic_ftp2 start"
$ns at 4.0 "$traffic_ftp2 stop"
$ns at 5.0 "finish"
$ns run
```

Sample I/O:



Assignment-14 : Congestion Control

Objective: To Simulate different Reno and Tahoe protocols and Plot the congestion graph in ns2.

CODE:

.tcl file

```
#Create a simulator object
set ns [new Simulator]
$ns rtproto LS

# Opening NAM and trace file
set namfile [open out.nam w]
$ns namtrace-all $namfile
set tracefile [open out.tr w]
$ns trace-all $tracefile

# Finish procedure
proc finish {} {
    global ns namfile tracefile
    $ns flush-trace
    #Close the NAM trace file
    close $namfile
    close $tracefile
    #Execute NAM on the trace file
    exec nam out.nam &
    exit 0
}

#Define different colors for data flows (for NAM)
$ns color 0 Red
$ns color 1 Green

# Create nodes
set node0 [$ns node]
set node1 [$ns node]
set node2 [$ns node]
set node3 [$ns node]
```

```

set node4 [$ns node]
set node5 [$ns node]

# Create links between nodes
$ns duplex-link $node0 $node2 10Mb 10ms DropTail
$ns duplex-link $node1 $node2 10Mb 10ms DropTail
$ns duplex-link $node2 $node3 10Mb 10ms DropTail
$ns duplex-link $node3 $node4 10Mb 10ms DropTail
$ns duplex-link $node3 $node5 10Mb 10ms DropTail

# The queue size at $R is to be 7, including the packet being sent
$ns queue-limit $node2 $node3 7

# Orient Links
$ns duplex-link-op $node0 $node2 orient right-down
$ns duplex-link-op $node1 $node2 orient right-up
$ns duplex-link-op $node2 $node3 orient right
$ns duplex-link-op $node3 $node4 orient right-up
$ns duplex-link-op $node3 $node5 orient right-down
$ns duplex-link-op $node2 $node3 queuePos 0.5

# Creating a TCP sender Tahoe and attach to node 0 (default)
set tcptahoe [new Agent/TCP]
# Creating a TCP sender Reno and attach to node 1
set tcpreno [new Agent/TCP/Reno]
# Setting flow
$tcptahoe set class_ 0
$tcptahoe set window_ 100
$tcptahoe set packetSize_ 800
$tcpreno set class_ 1
$tcpreno set window_ 100
$tcpreno set packetSize_ 800
$ns attach-agent $node0 $tcptahoe
$ns attach-agent $node1 $tcpreno

# Trace variables
$tcptahoe attach $tracefile
$tcptahoe tracevar cwnd_
$tcptahoe tracevar ssthresh_
$tcptahoe tracevar ack_
$tcptahoe tracevar maxseq_

#Create a TCP receive agent (a traffic sink) and attach it to B

```

```

set endtahoe [new Agent/TCPSink]
$ns attach-agent $node4 $endtahoe

set endreno [new Agent/TCPSink]
$ns attach-agent $node5 $endreno

#Connect the traffic source with the traffic sink
$ns connect $tcptahoe $endtahoe
$ns connect $tcpreno $endreno

#Schedule the connection data flow; start sending data at T=0, stop at T=10.0
set ftptahoe [new Application/FTP]
$ftptahoe attach-agent $tcptahoe
$ns at 0.0 "$ftptahoe start"
$ns at 10.0 "finish"

set ftpreno [new Application/FTP]
$ftpreno attach-agent $tcpreno
$ns at 0.0 "$ftpreno start"
$ns at 10.0 "finish1"

#Plot Congestion Window Graph
proc plotWindow {tcpSource outfile} {
    global ns
    set now [$ns now]
    set cwnd [$tcpSource set cwnd_]

    # the data is recorded in a file called congestion.xg (this can be plotted #
    using xgraph or gnuplot. this example uses xgraph to plot the cwnd_
    puts $outfile "$now $cwnd"
    $ns at [expr $now+0.1] "plotWindow $tcpSource $outfile"
}

set outfile [open "congestiontahoe.xg" w]
$ns color 0 Green
$ns at 0.0 "plotWindow $tcptahoe $outfile"
proc finish_cong {} {
    exec xgraph congestion.xg -geometry 300x300 &
    exit 0
}

set outfile [open "congestionreno.xg" w]

```

```

$ns color 1 Purple
$ns at 0.0 "plotWindow $tcpreno $outfile"
proc finish_cong {} {
    exec xgraph congestion.xg -geometry 300x300 &
    exit 0
}

#Run the simulation
$ns run

```

Throughput.py

```

#!/usr/bin/python3

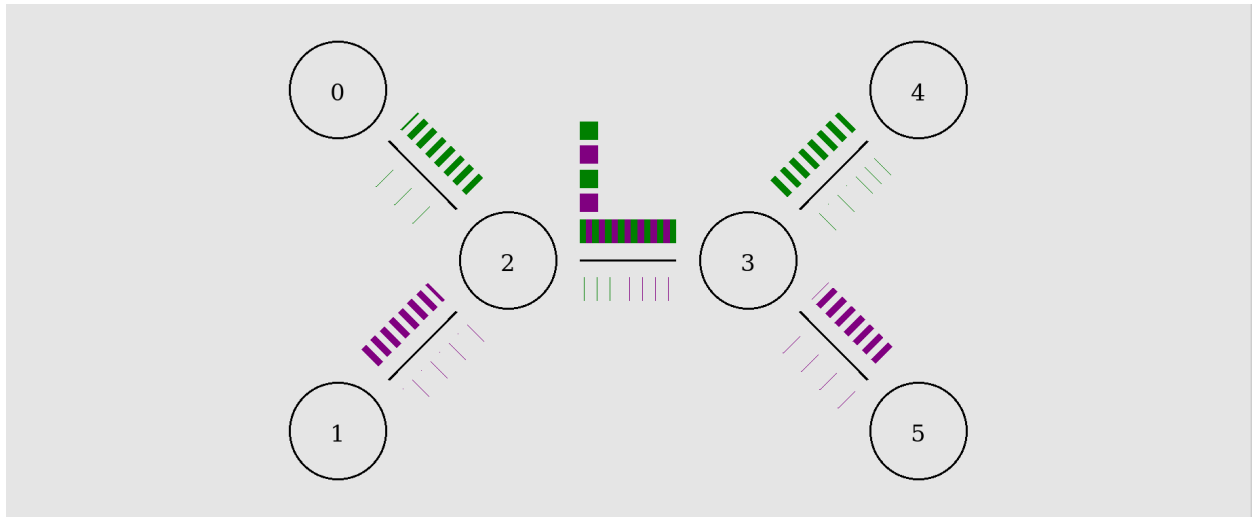
f1 = open("out.tr","r")
r1 = f1.readlines()

r=0
for val in r1:
    if val[0]=="r":
        li = list(val.split(" "))
        if li[4]=="tcp":
            r+=int(li[5])
            r-=(int(li[5])%512)

f1.close()
t = ((r/10)*(8/1000))/1000
print("Throughput = ",t,"mbps")

```

Sample I/O:



```
root@OSNPL-C6: ~/Desktop/NetworksLAB/14/Final Correct
root@OSNPL-C6:~/Desktop/NetworksLAB/14/Final Correct# python3 throughput.py
Throughput = 7.4440704 mbps
root@OSNPL-C6:~/Desktop/NetworksLAB/14/Final Correct#
```

Congestion Graph Output:

