

# **LAB MANUAL**

## **CS334 NETWORK PROGRAMMING LAB**

**Ex.No.1**

**Familiarization of various system calls used for OS and network programming**

**OBJECTIVE**

**To understand the use and functioning of various system calls.**

**1 a) AIM: To write a program for Fork, Getpid system call using C**

**ALGORITHM**

**Step 1:** Start the program

**Step 2:** In main function create child process using fork() system call and store the return value in pid

**Step 3:** If pid < 0 then error in process creation

**Step 4:** If pid==0 then the process is child process and get process id using getpid() system call

**Step 5:** Else the process is parent and get process id using getpid() and display child process id using pid

**Step 6:** Execute the program

**PROGRAM**

```
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>
#include<stdlib.h>
main()
{
int pid;
pid=fork();
if(pid== -1)
{
```

```

//perror("Error in fork");
printf("Error in fork ");
exit(1);
}
else if(pid==0)
{
printf("Output comes from Child process");
printf("\nChild process ID = %d", getpid());
}
else
{
printf("\n Output comes from parent process");
printf("\n Parent process ID = %d,Child process ID=%d", getpid(), pid);
}
exit(0);
}

```

### **NEW PROGRAM**

```

#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>
#include<stdlib.h>
int main()
{
int pid;
pid=fork();
if(pid== -1)
{
perror("Error in fork");
exit(1);
}

```

```
}  
else if(pid==0)  
{  
printf("Output comes from Child process");  
printf("\nChild process ID = %d\n", getpid());  
execlp("/bin/ls","ls",NULL);  
}  
else  
{  
printf("\n parent process wait for child to complete");  
wait(NULL);  
printf("\nchild process completed\n");  
printf("\n parent process id = %d, child process id = %d",getpid(),pid());  
exit(0);  
}  
}
```

**Ex.No.1(b)**

## **EXEC SYSTEM CALL**

**AIM: To write a program to execute exec system call using C**

### **ALGORITHM**

**Step 1:** Start the program.

**Step 2:** Enter the option.

**Step 3:** If the option is 1 execute "ls" command using execlp( ) system call

**Step 4:** If the option is 2 execute "ps" command using execlp( ) system call

**Step 5:** If the option is 3 execute "who" command using execlp( ) system call

**Step 6:** If the option is 4 display "invalid choice"

**Step 7:** Execute the program.

### **PROGRAM**

// EXEC SYSTEM CALL

```
#include<stdio.h>
```

```
#include<unistd.h>
```

```
#include<sys/types.h>
```

```
main()
```

```
{
```

```
    int choice;
```

```
    do
```

```
    {
```

```
        printf("\nMain Menu");
```

```
        printf("\nSelect an option");
```

```
        printf("\n1.Execute ls command");
```

```
        printf("\n2.Execute ps command");
```

```
        printf("\n3.Execute who command");
```

```
        printf("\n4.Exit");
```

```
        printf("\nEnter your choice");
```

```
        scanf("%d",&choice);
```

```
switch(choice)
{
case 1:
    if(fork())
        wait(0);
    else
        execlp("ls","ls",(char*)NULL);
    break;
case 2:
    if(fork())
        wait(0);
    else
        execlp("ps","ps",(char*)NULL);
    break;
case 3:
    if(fork())
        wait(0);
    else
        execlp("who","who",(char*)NULL);
    break;
case 4:
    exit(0);
default:
    printf("Invalid choice");
}
}while(choice<=4);
}
```

**Ex.No.1(c)**

**WAIT SYSTEM CALL**

**AIM:** To write a program for wait system call using C

**ALGORITHM**

**Step 1:** Start the program.

**Step 2:** Define main function

**Step 3:** Create child process using fork system call

**Step 4:** Execute and display child process work ie list out the files

**Step 5:** Parent process wait for child process execution to be completed using wait ( ) system call

**Step 6:** execute and display parent process work ie process status

**Step 7:** Execute the program

**PROGRAM**

```
#include<unistd.h>
```

```
#include<stdio.h>
```

```
#include<sys/types.h>
```

```
#include<stdlib.h>
```

```
main()
```

```
{
```

```
int p,m,j;
```

```
p=fork();
```

```
if(p==0)
```

```
{
```

```
printf("\nChild Process\n");
```

```
execlp("ls","ls",NULL);
```

```
}
```

```
else
```

```
{  
wait(0);  
printf("\n Parent process Wait for child process execution");  
execlp("ps","ps",NULL);  
}  
}
```

## *Program #: 2*

### Process and Thread Programs

#### AIM

To implement programs related to process and thread.

#### **Process Creation Concepts**

Processes are the primitive units for allocation of system resources. Each process has its own **address space** and (usually) one thread of control. A process executes a program; you can have multiple processes executing the same program, but each process has its own copy of the program within its own address space and executes it independently of the other copies.

Processes are organized **hierarchically**. Each process has a **parent process**, which explicitly arranged to create it. The processes created by a given parent are called its **child processes**. A child inherits many of its attributes from the parent process.



A **process ID number** names each process. A unique process ID is allocated to each process when it is created. The lifetime of a process ends when its termination is reported to its parent process; at that time, all of the process resources, including its process ID, are freed.

Processes are created with the **fork()** system call (so the operation of creating a new process is sometimes called **forking a process**). The child process created by fork is a copy of the original parent process, except that it has its own process ID.

After forking a child process, both the parent and child processes continue to execute normally. If you want your program to wait for a child process to finish executing before continuing, you must do this explicitly after the fork operation, by calling **wait()** or **waitpid()**. These functions give you limited information about why the child terminated--for example, its exit status code.

A newly forked child process continues to execute the same program as its parent process, at the point where the fork call returns. You can use the return value from fork to tell whether the program is running in the parent process or the child process.

When a child process terminates, its death is communicated to its parent so that the parent may take some appropriate action. A process that is waiting for its parent to accept its return code is called a **zombie process**. If a parent dies before its child, the child (**orphan process**) is automatically adopted by the original “**init**” process whose PID is **1**.

## **PROGRAM**

```
class Count extends Thread
{
    Count()
    {
        super("my extending thread");
        System.out.println("my thread created" + this);
        start();
    }
    public void run()
    {
        try
        {
            for (int i=0 ;i<10;i++)
            {
                System.out.println("Printing the count " + i);
                Thread.sleep(1000);
            }
        }
        catch (InterruptedException e)
        {
            System.out.println("my thread interrupted");
        }
        System.out.println("My thread run is over" );
    }
}
```

```

    }
}
class ExtendingExample
{
    public static void main(String args[])
    {
        Count cnt = new Count();
        try
        {
            while(cnt.isAlive())
            {
                System.out.println("Main thread will be alive till the child thread
is live");
                Thread.sleep(1500);
            }
        }
        catch(InterruptedException e)
        {
            System.out.println("Main thread interrupted");
        }
        System.out.println("Main thread's run is over" );
    }
}

```

## Inter process communication

### AIM

To study and implement IPC using:

a) PIPE,

### **PROGRAM**

IPC using PIPES

```
#include <iostream>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
using namespace std;
int main()
{
    int ret_val;
    int fd[2];
    char buff[100]="";
    char string1[100]="";
    cout<<"\nEnter the msg= ";
    cin>>string1;
    ret_val = pipe(fd);
    if (ret_val != 0)
    {
        cout<<"Unable to create a pipe";
        exit(1);
    }
    if (fork() == 0)
    {
        close(fd[0]);
        ret_val = write(fd[1],string1,strlen(string1));
        if (ret_val != strlen(string1))
        {
            cout<<"Write did not return expected value\n";
            exit(2);
        }
    }
    else
    {
        close(fd[1]);
        ret_val = read(fd[0],buff,strlen(string1));
        if (ret_val != strlen(string1))
```

```
{  
    cout<<"Read did not return expected value\n";  
    exit(3);  
}  
    cout<<"Parent read "<<buff<<" from the child program\n";  
}  
exit(0);  
}
```

### ***PROGRAM 3***

## **Program #: 5**

### **Socket Programming using TCP**

#### **AIM**

To write a java program to implement unidirectional chat.

#### **ALGORITHM**

##### **Server**

1. Start
2. Create a server socket and a simple socket.
3. Using accept() ,accept client socket.
4. Wait for client request and establish the TCP connection with client bound to a port number
5. Read the client data into a buffer.
6. Store data to a string and display the data
7. Check whether the data is “quit”, if matching then goto step 8else goto step 5.
8. Stop

##### **Client**

1. Start
2. Create a socket.
3. Establish the TCP connection with server.
4. Read the keyboard input data from server into a buffer.
5. Store the data into a string and send the data to the client via outputstream .
6. Check whether the data is "quit", if matching then goto step 7 else goto step4.
7. Stop

#### **PROGRAM**

##### **Server**

```
import java.io.*;
import java.net.*;
public class userver
{
    public static void main(String args[])throws Exception
    {
        String str;
        try {
            ServerSocketss = new ServerSocket(4000);
            Socket s;
            s=ss.accept();
            System.out.println("Connection Established");
            BufferedReaderbr=new BufferedReader(new InputStreamReader(s.getInputStream()));
            while(true)
```

```

{
str=br.readLine();
If(str.equals("exit"))
{
System.exit(0);

}
System.out.println(" "+str);
}
}
catch(Exception e)
{
}

}
}

```

### **Client**

```

import java.io.*;
import java.net.*;
public class uclient
{
public static void main(String args[])
{
String str;
try{
Socket s=new Socket("localhost",4000);
BufferedReaderbr=new BufferedReader(new InputStreamReader(System.in));
PrintWriter pw=new PrintWriter(s.getOutputStream(),true);
while(true)

{
str=br.readLine();
if(str.equals("exit"))
{
System.exit(0);
}
pw.println(str);
}
}
catch(Exception e)

{
}
}
}
}

```

## OUTPUT

### Server

```
Microsoft Windows [Version 6.1.7600]  
Copyright (c) 2009 Microsoft Corporation. All rights reserved.  
  
D:\>start  
  
D:\>javac userver.java  
  
D:\>java userver  
Connection Established  
hi  
hello  
bye
```

### Client



```
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

D:\>javac uclient.java
D:\>java uclient
hi
hello
bye
```

### *Program #: 6*

## Socket Programming using UDP

### AIM

To Implement client server communication using UDP.

### ALGORITHM

#### **ALGORITHM**

##### **Server**

Define a class Userver

1. Declare a static DatagramSocket object ds
2. Declare a static DatagramPacket object dp5, dp3
3. Define a static Listener class that implements Runnable interface
  - 3.1 Create a new instance of the Datagram Socket object ds
  - 3.2 Create an object ip of InetAddress class and call getLocalHost() method to get the IP address of Local host.
  - 3.3 Create an object br of BufferedReader class and connect it with System.in to read input from keyboard

- 3.4 Repeat steps 3.5 to 3.9 until String read is not quit
- 3.5 Read string from br to a variable s
- 3.6 Declare a new byte array bte of length of the read input string
- 3.7 Convert the string into byte data
- 3.8 Create a new instance of dp3 object with proper parameters
- 3.9 send the packet dp3
4. Define a static class writer that implements Runnable interface
- 4.1 Create a new instance of the object with parameter port no. 27015
- 4.2 Create an object ip of Inet Address class and initialize it
- 4.3 Repeat steps 4.4 to 4.8 till received string is not quit
- 4.4 Declare a new byte array
- 4.5 Create a new instance of the DatagramPacket object dp5 with proper parameters
- 4.6 Receive dp5 through socket
- 4.7 Declare a string variable s with the value obtained from converting the received byte array
- 4.8 print string s
5. stop

Main function

1. Start threads writer and listener
2. stop

Define class uclient

1. Declare a DatagramSocket object ds
2. Declare static DatagramPacket dp1 and dp2
3. Define a static listener class that implements Runnable interface
- 3.1 Create a new instance of the datagram socket object ds
- 3.2 create and initialize object ip of InetAddress with local host
- 3.3 Create and object br of BufferedReader to attach with System.in
- 3.4 Repeat steps 3.5 to 3.8
- 3.5 Read a string to s
- 3.6 convert that string to byte array
- 3.7 Create a datagram packet dp2 from this array
- 3.8 send it
4. Create a new instance Datagram Socket with port no 27016

5. Create objects to receive data from socket and print that string until the entered string is quit
6. stop

## **ALGORITHM / PROCEDURE**

### **SERVER**

```
#include<stdio.h>
#include<netinet/in.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netdb.h>
#include<string.h>
#include<stdlib.h>
#define MAX 80
#define PORT 43454
#define SA struct sockaddr
void func(int sockfd)
{
    char buff[MAX];
    int n,clen;
    struct sockaddr_in cli;
    clen=sizeof(cli);
    for(;;)
    {
        bzero(buff,MAX);
        recvfrom(sockfd,buff,sizeof(buff),0,(SA *)&cli,&clen);
        printf("From client %s To client",buff);
        bzero(buff,MAX);
        n=0;
        while((buff[n++]=getchar())!='\n');
        sendto(sockfd,buff,sizeof(buff),0,(SA *)&cli,clen);
        if(strncmp("exit",buff,4)==0)
        {
            printf("Server Exit...\n");
            break;
        }
    }
}

int main()
{
    int sockfd;
    struct sockaddr_in servaddr;
    sockfd=socket(AF_INET,SOCK_DGRAM,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");
        func(sockfd);
        close(sockfd);
    }

```

## CLIENT

```

#include<sys/socket.h>
#include<netdb.h>
#include<string.h>
#include<stdlib.h>
#include<stdio.h>
#define MAX 80
#define PORT 43454
#define SA struct sockaddr
int main()
{
    char buff[MAX];
    int sockfd,len,n;
    struct sockaddr_in servaddr;
    sockfd=socket(AF_INET,SOCK_DGRAM,0);
    if(sockfd==-1)
    {
        printf("socket creation failed...\n");
    }

```

```

        exit(0);
    }
    else
    printf("Socket successfully created..\n");
    bzero(&servaddr,sizeof(len));
    servaddr.sin_family=AF_INET;
    servaddr.sin_addr.s_addr=inet_addr("127.0.0.1");
    servaddr.sin_port=htons(PORT);
    len=sizeof(servaddr);
    for(;;)
    {
    printf("\nEnter string : ");
    n=0;
    while((buff[n++]=getchar())!='\n');
    sendto(sockfd,buff,sizeof(buff),0,(SA *)&servaddr,len);
    bzero(buff,sizeof(buff));
    recvfrom(sockfd,buff,sizeof(buff),0,(SA *)&servaddr,&len);
    printf("From Server : %s\n",buff);
    if(strncmp("exit",buff,4)==0)
    {
    printf("Client Exit...\n");
break;
    }
    }
    close(sockfd);
}

```

## OUTPUT

### SERVER SIDE

\$ cc udpchatserver.c

\$ ./a.out

Socket successfully created..

Socket successfully binded..

From client hai

To client hello

From client exit

To client exit

Server Exit...

\$

### CLIENT SIDE

```
$ cc udpchatclient.c
$ ./a.out
    Socket successfully created..
    Enter string : hai
From Server : hello
    Enter string : exit
From Server : exit
    Client Exit...
$
```

## **Program #: 7**

### **Implementation of multi user chat server using TCP\_**

#### **AIM**

To write a java program to implement central server chat.

#### **ALGORITHM**

##### **Server**

- Step 1: Start
- Step 2: Create a server socket and two simple socket
- Step 3: Create two thread t1 and t2
- Step 4: Establish TCP connection with each client
- Step 5: Check whether the current active thread is t1 then goto step 6 else goto step 7
- Step 6: Read the data from buffer of client 1 and store in the buffer of client 2
- Step 7: Read the data from buffer of client 2 and store in the buffer of client 1
- Step 8: Check whether the data is "exit" if matching goto step 9 else goto step 5
- Step 9: Create an object for the class and invoke the threads in the main() method
- Step 6: Stop

##### **Client**

- Step 1: Start
- Step 2: Create a socket
- Step 3: Create two thread t1 and t2
- Step 4: Establish TCP connection with server
- Step 5: Check whether the current active thread is t1 then goto step 6 else goto step 7
- Step 6: Store the data from buffer to a string and display the data
- Step 7: Read the keyboard input data to the buffer
- Step 8: Check whether the data is "exit" if matching goto step 9 else goto step 5
- Step 9: Create an object for the class and invoke the threads in the main() method

Step 6: Stop

#### **PROGRAM**

##### **Server**

```
import java.net.*;
import java.io.*;
public class cserver implements Runnable
{
    ServerSocket ss;
    BufferedReader br1,br2;
    PrintWriter pw1,pw2;
    String str;
    Socket s1,s2;
    Thread t1=null,t2=null;
    cserver()
    {try{
```



```

ss=new ServerSocket(5000);
s1=ss.accept();
System.out.println("client 1 connected");
s2=ss.accept();
System.out.println("client 2 connected");
}
catch(Exception e){}
}
public void run()
{
try
{
do
{
if(Thread.currentThread()==t1)
{
br1=new BufferedReader(new InputStreamReader(s1.getInputStream()));
pw1=new PrintWriter(s2.getOutputStream(),true);

str=br1.readLine();
pw1.println(str);
}
else
{
br2=new BufferedReader(new InputStreamReader(s2.getInputStream()));
pw2=new PrintWriter(s1.getOutputStream(),true);
str=br2.readLine();
pw2.println(str);
}
}while(!str.equals("exit"));
}
catch(Exception e){}
}
public static void main(String args[])throws Exception
{
cserver ob=new cserver();
ob.t1=new Thread(ob);
ob.t2=new Thread(ob);
ob.t1.start();
ob.t2.start();

}
}

```

### **Client 1**

```

import java.net.*;
import java.io.*;
public class cclient1 implements Runnable
{

```

```

Socket s;
BufferedReader br1,br2;
PrintWriter pw;
String str;
Thread t1,t2;
cclient1()
{
try{
s=new Socket("localhost",5000);
br1=new BufferedReader(new InputStreamReader(System.in));
br2=new BufferedReader(new InputStreamReader(s.getInputStream()));
pw=new PrintWriter(s.getOutputStream(),true);
}
catch(Exception e){}
}
public void run()
{
try
{
do
{
if(Thread.currentThread()==t1)
{
str=br1.readLine();
pw.println(str);
}
else
{
str=br2.readLine();
System.out.println("client 2 says:"+str);

}
}while(!str.equals("exit"));
}
catch(Exception e){}
}
public static void main(String args[])throws Exception
{
cclient1 ob=new cclient1();
ob.t1=new Thread(ob);
ob.t2=new Thread(ob);
ob.t1.start();

ob.t2.start();
}
}

```

## **Client 2**

```

import java.net.*;
import java.io.*;

```

```

public class cclient2 implements Runnable
{
    Socket s;
    BufferedReader br1,br2;
    PrintWriter pw;
    String str;
    Thread t1,t2;
    cclient2()
    {
        try{
            s=new Socket("localhost",5000);
            br1=new BufferedReader(new InputStreamReader(System.in));
            br2=new BufferedReader(new InputStreamReader(s.getInputStream()));
            pw=new PrintWriter(s.getOutputStream(),true);
        }
        catch(Exception e){}
    }
    public void run()
    {
        try
        {
            do
            {
                if(Thread.currentThread()==t1)

                {
                    str=br2.readLine();
                    System.out.println("client 1 says:"str); }
                else
                {
                    str=br1.readLine();
                    pw.println(str);
                }
            }while(!str.equals("exit"));
        }
        catch(Exception e){}
    }
    public static void main(String args[])throws Exception
    {cclient2 ob=new cclient2();
    ob.t1=new Thread(ob);
    ob.t2=new Thread(ob);
    ob.t1.start();
    ob.t2.start();
    }
}

```

**OUTPUT**  
**Server**

```
C:\Windows\System32\cmd.exe - java cserver
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

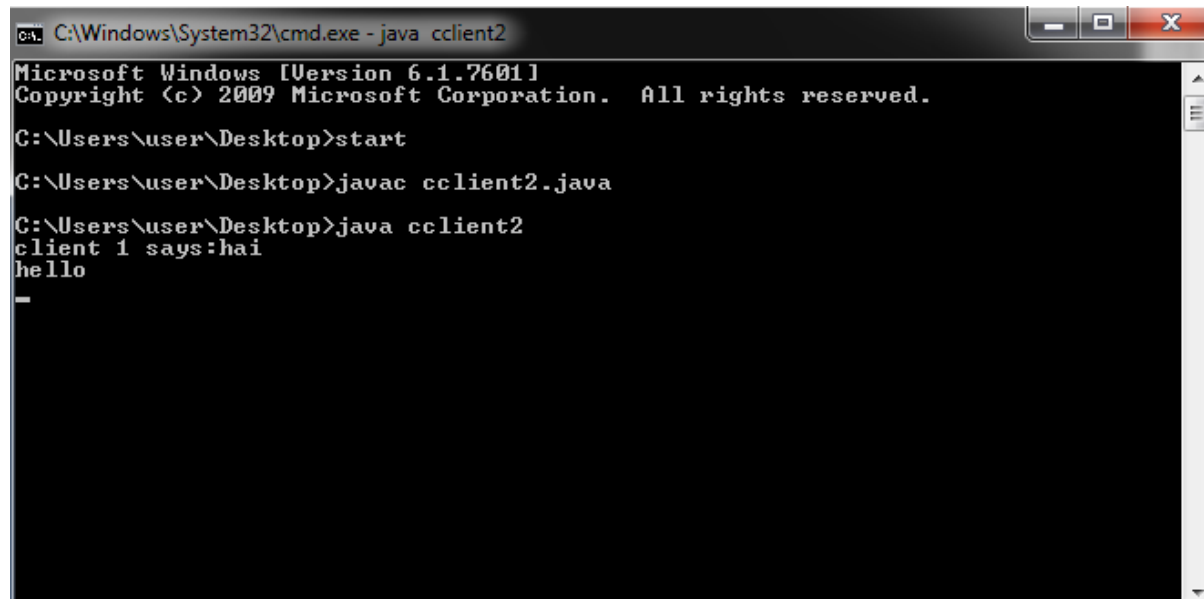
C:\Users\user\Desktop>javac cserver.java
C:\Users\user\Desktop>java cserver
client 1 connected
client 2 connected
```

## Client1

```
C:\Windows\System32\cmd.exe - java cclient1
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\user\Desktop>start
C:\Users\user\Desktop>javac cclient1.java
C:\Users\user\Desktop>java cclient1
hai
client 2 says:hello
```

## Client2



```
C:\Windows\System32\cmd.exe - java cclient2
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\user\Desktop>start
C:\Users\user\Desktop>javac cclient2.java
C:\Users\user\Desktop>java cclient2
client 1 says: hai
hello
-
```

## ***Program #: 8***

### **Implementation of concurrent time server application using UDP**

#### **AIM**

To implement concurrent time server application using UDP.

#### **PROGRAM**

##### **Client**

```
import java.net.*;
import java.io.*;
class udpclient{
    public static void main (String args[])throws Exception{
        DatagramSocket dsoc=new DatagramSocket(6244);
        byte buff[]=new byte[1024];
        DatagramPacket dpak=new DatagramPacket (buff,buff.length);
        Dsoc.receive(dpak);
        System.out.println(new String(dpak.getData()))
    }
}
```

##### **Server**

```
import java.net.*;
import java.io.*;
class udpserver{
    public static void main (String args[])throws Exception{
        DatagramSocket dsoc=new DatagramSocket(5217);
```

```
InetAddress host=InetAddress.getLocal.Host();  
String str=(new Date of()  
byte buff[]=str.getBytes();  
dsoc.send(new Datagram Packet(buff, buff.length,host(6244)));  
dsoc.close()  
}  
}
```

## **Program #:12**

### **Design and configuration of a network with multiple subnets**

#### **AIM**

To design and configure a network with multiple subnets.

#### **ALGORITHM / PROCEDURE**

Network simulator is a piece of software or hardware that predicates the behavior of a network without an actual network being present. NS2 is a simulation tool for networks. It supports several algorithms for routing and queuing. It can set up packet traffic similar to internet and measure various parameters. NS2 is very useful because it is very expensive to check feasibility of new algorithms, check architectures, check topologies etc. network simulator is a name for series of discrete event network simulator. Simulators are used in the simulation of routing protocols, and are heavily used in ad-hoc networking research, and support popular network protocols, offering simulation results for wired and wireless networks alike.

#### **Features**

It can be employed in most Unix systems and Windows. Most of NS2 code is in C++. It uses TCL as its scripting language, OTCL adds object orientation to TCL. NS (version 2) is an object oriented, discrete event driven network simulator that is freely distributed and open source.

- Protocols: TCP, UDP, HTTP, Routing algorithms, MAC etc
- Traffic Models: CBR, VBR, Web etc
- Error Models: Uniform, bursty etc
- Misc: Radio propagation, Mobility models, Energy Models
- Topology Generation tools
- Visualization tools (NAM), Tracing

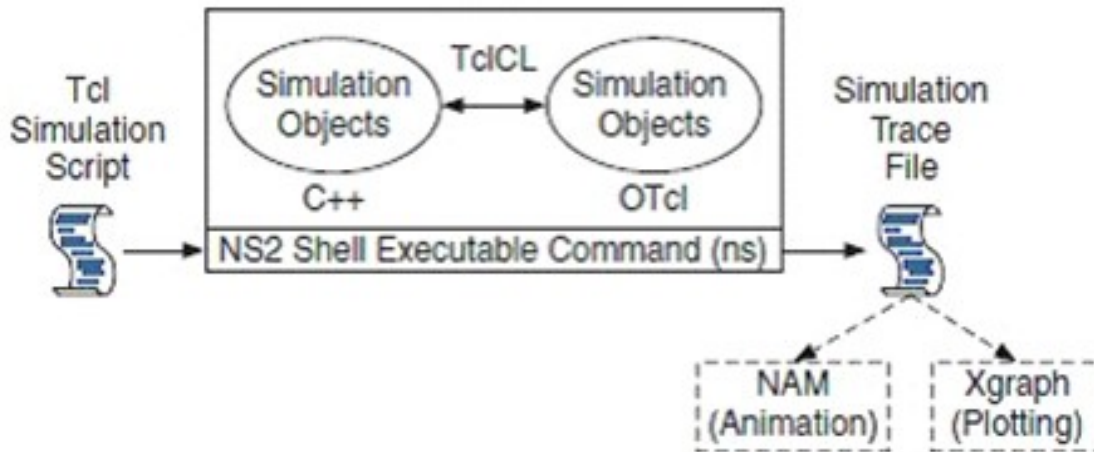
#### **NS Structure**

- NS is an object oriented discrete event simulator
  - Simulator maintains list of events and executes one event after another
  - Single thread of control: no locking or race conditions
- Back end is C++ event scheduler
  - Protocols mostly
  - Fast to run, more control



- Front end is OTCL
  - Creating scenarios, extensions to C++ protocols
  - Fast to write and change

## NS Basic Architecture



## Platforms

It can be employed in most Unix systems (FreeBSD, Linux, Solaris) and Windows.

## Source code

Most of NS2 code is in C++

## Scripting language

It uses TCL as its scripting language OTcl adds object orientation to TCL. NS (version 2) is an object-oriented, discrete event driven network simulator developed at UC Berkely written in C++ and OTcl. NS is popularly used in the simulation of routing and multicast protocols, among others, and is heavily used in ad-hoc networking research. NS supports an array of popular network protocols, offering simulation results for wired and wireless networks alike. It can be also used as limited-functionality network emulator. It is popular in academia for its extensibility (due to its open source model) and plentiful online documentation. NS was built in C++ and provides a simulation interface through OTcl, an object-oriented dialect of Tcl. The user describes a network topology by writing OTcl scripts, and then the main NS program simulates that topology with specified parameters.

## Protocols implemented in NS2

Transport layer (Traffic Agent) – TCP, UDP

Network layer (Routing agent)

Interface queue – FIFO queue, Drop Tail queue, Priority queue

Logic link control layer – IEEE 802.2, AR

### **How to use NS2**

Design Simulation – Determine simulation scenario

Build ns-2 script using tcl.

Run simulation

### **Simulation with NS2**

- Define objects of simulation.
- Connect the objects to each other
- Start the source applications. Packets are then created and are transmitted through network.
- Exit the simulator after a certain fixed time.

### **Accompanying Tools**

#### **nam**

A tool that allows visualizing the motion of packets through the nodes and links of the network. It can either start nam with the command “nam <nam-file>”, where “<namfile>” is the name of a nam trace file that was generated by ns, or can execute it directly out of the Tcl simulation script for the simulation which want to visualize.

#### **Xgraph**

A tool that allows to plot the results of the simulation in the form of curves. xgraph is a plotting program which can be used to create graphic representations of simulation results. It can create output files in Tcl scripts, which can be used as data sets for xgraph. Call xgraph to display the results with the command “xgraph <data-file>”.

### **NS programming Structure**

- Create the event scheduler
- Turn on tracing
- Create network topology
- Create transport connections
- Generate traffic
- Insert errors

#### **Creating Event Scheduler**

- Create event scheduler: set ns [new Simulator]
  - Schedule an event: \$ns at <time> <event>
- event is any legitimate ns/tcl function

```

$ns at 5.0 "finish"

proc finish {} {
    global ns nf
    close $nf
    exec nam out.nam &
    exit 0
}

```

- Start Scheduler

```
$ns run
```

## **Tracing**

- All packet trace

```

$ns trace-all [open out.tr w]

<event> <time> <from> <to> <pkt> <size>

...

<flowid> <src> <dst> <seqno> <aseqno>

+ 0.51 0 1 cbr 500 ----- 0 0.0 1.0 0 2
_ 0.51 0 1 cbr 500 ----- 0 0.0 1.0 0 2
r 0.514 0 1 cbr 500 ----- 0 0.0 1.0 0 0

```

- Variable trace

```

set par [open output/param.tr w]

$tcp attach $par

$tcp trace cwnd_

$tcp trace maxseq_

$tcp trace rtt_

```

## **Tracing and Animation**

- Network Animator

```
set nf [open out.nam w]
```

```

$ns namtrace-all $nf

proc finish {} {

global ns nf

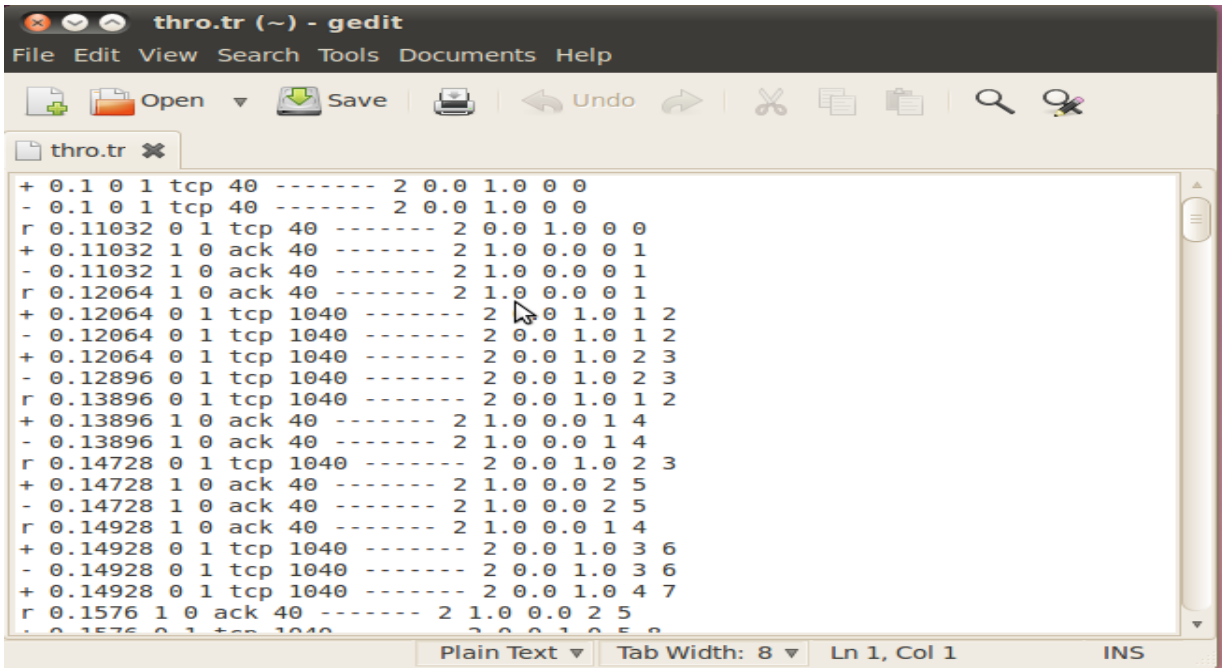
close $nf

exec nam out.nam &

exit 0

}

```



The screenshot shows a gedit window titled 'thro.tr (~) - gedit'. The window contains a network trace log with columns for time, IP addresses, protocol, sequence numbers, and various statistics. The log shows a series of TCP connections and acknowledgments between two hosts. The status bar at the bottom indicates 'Plain Text', 'Tab Width: 8', and 'Ln 1, Col 1'.

```

+ 0.1 0 1 tcp 40 ----- 2 0.0 1.0 0 0
- 0.1 0 1 tcp 40 ----- 2 0.0 1.0 0 0
r 0.11032 0 1 tcp 40 ----- 2 0.0 1.0 0 0
+ 0.11032 1 0 ack 40 ----- 2 1.0 0.0 0 1
- 0.11032 1 0 ack 40 ----- 2 1.0 0.0 0 1
r 0.12064 1 0 ack 40 ----- 2 1.0 0.0 0 1
+ 0.12064 0 1 tcp 1040 ----- 2 0.0 1.0 1 2
- 0.12064 0 1 tcp 1040 ----- 2 0.0 1.0 1 2
+ 0.12064 0 1 tcp 1040 ----- 2 0.0 1.0 2 3
- 0.12896 0 1 tcp 1040 ----- 2 0.0 1.0 2 3
r 0.13896 0 1 tcp 1040 ----- 2 0.0 1.0 1 2
+ 0.13896 1 0 ack 40 ----- 2 1.0 0.0 1 4
- 0.13896 1 0 ack 40 ----- 2 1.0 0.0 1 4
r 0.14728 0 1 tcp 1040 ----- 2 0.0 1.0 2 3
+ 0.14728 1 0 ack 40 ----- 2 1.0 0.0 2 5
- 0.14728 1 0 ack 40 ----- 2 1.0 0.0 2 5
r 0.14928 1 0 ack 40 ----- 2 1.0 0.0 1 4
+ 0.14928 0 1 tcp 1040 ----- 2 0.0 1.0 3 6
- 0.14928 0 1 tcp 1040 ----- 2 0.0 1.0 3 6
+ 0.14928 0 1 tcp 1040 ----- 2 0.0 1.0 4 7
r 0.1576 1 0 ack 40 ----- 2 1.0 0.0 2 5
- 0.1576 0 1 tcp 1040 ----- 2 0.0 1.0 5 0

```

### Creating topology

- Two nodes connected by a link
- Creating nodes

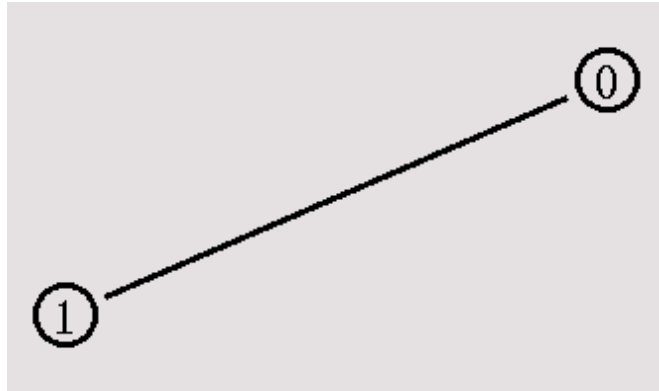
```
set n0 [$ns node]
```

```
set n1 [$ns node]
```

- Creating link between nodes

```
$ns <link_type> $n0 $n1 <bandwidth> <delay><queue-type>
```

```
$ns duplex-link $n0 $n1 1Mb 10ms DropTail
```



### **Sending data**

- Create UDP agent

```
set udp0 [new Agent/UDP]  
$ns attach-agent $n0 $udp0
```

- Create CBR traffic source for feeding into UDP agent

```
set cbr0 [new Application/Traffic/CBR]  
$cbr0 set packetSize_ 500  
$cbr0 set interval_ 0.005  
$cbr0 attach-agent $udp0
```

- Create traffic sink

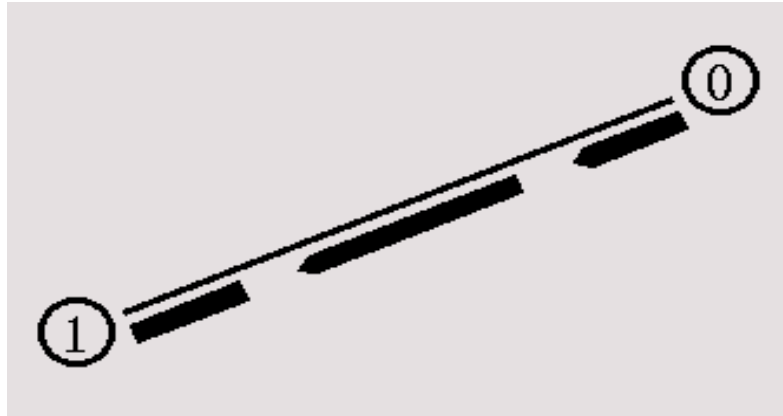
```
set null0 [new Agent/Null]  
$ns attach-agent $n1 $null0
```

- Connect two agents

```
$ns connect $udp0 $null0
```

- Start and stop of data

```
$ns at 0.5 "$cbr0 start"  
$ns at 4.5 "$cbr0 stop"
```



### **Creating TCP Connections**

- Create TCP agent and attach it to the node  

```
set tcp0 [new Agent/TCP]  
$ns attach-agent $n0 $tcp0
```
- Create a Null Agent and attach it to the node  

```
set null0 [new Agent/TCPSink]  
$ns attach-agent $n1 $null0
```
- Connect the agents  

```
$ns connect $tcp0 $null0
```

### **Traffic on top of TCP**

- FTP  

```
set ftp [new Application/FTP]  
$ftp attach-agent $tcp0
```
- Telnet  

```
set telnet [new Application/Telnet]  
$telnet attach-agent $tcp0
```

### **PROCEDURE**

STEP 1: Start

STEP 2: Create the simulator object ns for designing the given simulation

STEP 3: Open the trace file and nam file in the write mode

STEP 4: Create the nodes of the simulation using the 'set' command

STEP 5: Create links to the appropriate nodes using \$ns duplex-link command

STEP 6: Set the orientation for the nodes in the simulation using 'orient' command

STEP 7: Create TCP agent for the nodes and attach these agents to the nodes

STEP 8: The traffic generator used is FTP for both node0 and node1

STEP 9: Configure node1 as the sink and attach it

STEP10: Connect node0 and node1 using 'connect' command

STEP 11: Setting color for the nodes

STEP 12: Schedule the events for FTP agent 10 sec

STEP 13: Schedule the simulation for 5 minutes

### **PROGRAM**

```
# Create a new instance of the Simulator
set ns [new Simulator]

# Open trace file for analysis
set tf [open 2_tcpsemd.tr w]
$ns trace-all $tf

# Open namtrace file for visualization
set nf [open 2_tcpsemd.nam w]
$ns namtrace-all $nf

#Assign ids to colors
$ns color 0 Blue

# Create two nodes
set n0 [$ns node]
set n1 [$ns node]

# Set up duplex links between the nodes
$ns duplex-link $n0 $n1 1Mb 10ms DropTail

# Set n0 as TCP Source
set tcp [new Agent/TCP]
$ns attach-agent $n0 $tcp
```

```
# Set n1 as TCP Sink
set sink [new Agent/TCPSink]
$ns attach-agent $n1 $sink

# Connect TCP instance with Sink instance
$ns connect $tcp $sink

# Setting Color to node
$tcp set fid_ 0

# Set FTP Application on n0:
set ftp [new Application/FTP]
$ftp set packetSize_ 500
$ftp set interval_ 0.005
$ftp attach-agent $tcp

# Start and Stop FTP Application
$ns at 0.5 "$ftp start"
$ns at 4.5 "$ftp stop"

# Define the finish procedure
proc finish {} {
    global ns tf nf
    $ns flush-trace
    close $tf
    close $nf
    exec nam 2 _tcp.send.nam &
    exit 0
}

# Terminate the simulation
$ns at 5.0 "finish"
$ns run
```



## OUTPUT

