**Ex No : 5 Daemon Program**

**AIM:**

To create a Daemon Program

**ALGORITHM:**

* Create a Daemon Class
* Checking whether the thread is Daemon or not
* Setting user thread t1 to Daemon
* Starting first 2 threads
* Setting user thread t3 to Daemon

**Program:**

public class DaemonThread extends Thread{

public DaemonThread(String name){

super(name);

}

public void run()

{

if(Thread.currentThread().isDaemon())

{

System.out.println(getName() + " is Daemon thread");

}

else

{

System.out.println(getName() + " is User thread");

}

}

public static void main(String[] args)

{

DaemonThread t1 = new DaemonThread("t1");

DaemonThread t2 = new DaemonThread("t2");

DaemonThread t3 = new DaemonThread("t3");

t1.setDaemon(true);

t1.start();

t2.start();

t3.setDaemon(true);

t3.start();

}

}

**OUTPUT:**

t3 is Daemon thread

t1 is Daemon thread

t2 is User thread

**RESULT:**

The following daemon program is tested(three strings t1,t2&t3) and executed successfully.

**EXPERIMENT 6 - IMPLEMENTING STOP & WAIT PROTOCOL**

**AIM**

To implement the stop and wait protocol using java programming language.

PROCEDURE

**SERVER**

Step1: sequence ß 0

Step2: Accept new packet and assign sequence to it.

Step3: Send packet sequence with sequence number sequence.

Step4: Set timer for recently sent packets.

Step5: If error free acknowledgment from receiver and NextFrameExpected -> sequence

then sequenceß NextFrameExpected.

Step6: If time out then go to step3.

Step7: Stop.

**CLIENT**

Step1: Start.

Step2: NextFrameExpectedß 0, repeat steps 3 forever.

Step3: If error-free frame received and sequence= NextFrameExpected, then pass packet to

higher layer and NextFrameExpectedß NextFrameExpected+1(modulo 2).

Step4: Stop.

**PROGRAM**

**SERVER**

import java.io.\*;

import java.net.\*;

public class Sender{

Socket sender;

ObjectOutputStream out;

ObjectInputStream in;

String packet,ack,str, msg;

int n,i=0,sequence=0;

Sender(){}

public void run(){

try{

BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

System.out.println("Waiting for Connection....");

sender = new Socket("localhost",2005);

sequence=0;

out=new ObjectOutputStream(sender.getOutputStream());

out.flush();

in=new ObjectInputStream(sender.getInputStream());

str=(String)in.readObject();

System.out.println("reciver > "+str);

System.out.println("Enter the data to send....");

packet=br.readLine();

n=packet.length();

do{

try{

if(i<n){

msg=String.valueOf(sequence);

msg=msg.concat(packet.substring(i,i+1));

}else if(i==n){

msg="end";out.writeObject(msg);break;

}out.writeObject(msg);

sequence=(sequence==0)?1:0;

out.flush();

System.out.println("data sent>"+msg);

ack=(String)in.readObject();

System.out.println("waiting for ack.....\n\n");

if(ack.equals(String.valueOf(sequence))){

i++;

System.out.println("receiver > "+" packet recieved\n\n");

}else{

System.out.println("Time out resending data....\n\n");

sequence=(sequence==0)?1:0;

}}catch(Exception e){}

}while(i<n+1);

System.out.println("All data sent. exiting.");

}catch(Exception e){}

finally{

try{

in.close();

out.close();

sender.close();

}

catch(Exception e){}

}}

public static void main(String args[]){

Sender s=new Sender();

s.run();

}}

**CLIENT**

import java.io.\*;

import java.net.\*;

public class Receiver{

ServerSocket reciever;

Socket connection=null;

ObjectOutputStream out;

ObjectInputStream in;

String packet,ack,data="";

int i=0,sequence=0;

Receiver(){}

public void run(){

try{

BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

reciever = new ServerSocket(2005,10);

System.out.println("waiting for connection...");

connection=reciever.accept();

sequence=0;

System.out.println("Connection established :");

out=new ObjectOutputStream(connection.getOutputStream());

out.flush();

in=new ObjectInputStream(connection.getInputStream());

out.writeObject("connected .");

do{

try{

packet=(String)in.readObject();

if(Integer.valueOf(packet.substring(0,1))==sequence){

data+=packet.substring(1);

sequence=(sequence==0)?1:0;

System.out.println("\n\nreceiver >"+packet);

}

else

{

System.out.println("\n\nreceiver >"+packet +" duplicate data");

}if(i<3){

out.writeObject(String.valueOf(sequence));i++;

}else{

out.writeObject(String.valueOf((sequence+1)%2));

i=0;

}}

catch(Exception e){}

}while(!packet.equals("end"));

System.out.println("Data recived="+data);

out.writeObject("connection ended .");

}catch(Exception e){}

finally{

try{in.close();

out.close();

reciever.close();

}

catch(Exception e){}

}}

public static void main(String args[]){

Receiver s=new Receiver();

while(true){

s.run();

}

}

}

**RESULT:**

Thus the program is executed successfully and the output is verified

**EX.NO 7A- Program for Address Resolution Protocol (ARP) using TCP**

**Aim:**

To write a java program for simulating ARP protocols using TCP

**ALGORITHM:**

**Client**

1. Start the program

2. Using socket connection is established between client and server.

3. Get the IP address to be converted into MAC address.

4. Send this IP address to server.

5. Server returns the MAC address to client.

**Server**

1. Start the program

2. Accept the socket which is created by the client.

3. Server maintains the table in which IP and corresponding MAC addresses are stored.

4. Read the IP address which is send by the client.

5. Map the IP address with its MAC address and return the MAC address to client.

**Program**

**Client:**

import java.io.\*;

import java.net.\*;

import java.util.\*;

class Clientarp

{

public static void main(String args[])

{

try

{

BufferedReader in=new BufferedReader(new InputStreamReader(System.in));

Socket clsct=new Socket("127.0.0.1",5604);

DataInputStream din=new DataInputStream(clsct.getInputStream());

DataOutputStream dout=new

DataOutputStream(clsct.getOutputStream());

System.out.println("Enter the Logical address(IP):");

String str1=in.readLine();

dout.writeBytes(str1+'\n');

String str=din.readLine();

System.out.println("The Physical Address is: "+str);

clsct.close();

}

catch (Exception e)

{

System.out.println(e);

}

}

}

**Server:**

import java.io.\*;

import

java.net.\*;

import

java.util.\*;

class Serverarp

{

public static void main(String args[])

{

try

{

ServerSocket obj=new

ServerSocket(5604);

Socket obj1=obj.accept();

while(true)

{

DataInputStream din=new DataInputStream(obj1.getInputStream());

DataOutputStream dout=new

DataOutputStream(obj1.getOutputStream()); String str=din.readLine();

String ip[]={"165.165.80.80","165.165.79.1"};

String mac[]={"6A:08:AA:C2","8A:BC:E3:FA"};

for(int i=0;i<ip.length;i++)

{

if(str.equals(ip[i]))

{

dout.writeBytes(mac[i]+'\n');

break;

}

}

obj.close();

}

}

catch(Exception e)

{

System.out.println(e);

}

}

}

**Output:**

E:\networks>java Serverarp

E:\networks>java Clientarp

Enter the Logical address(IP):

165.165.80.80

The Physical Address is: 6A:08:AA:C2

**Result**

Thus the ARP protocol using TCP Sockets program was executed.

**EX.NO 7B Program for Reverse Address Resolution Protocol (RARP) using UDP**

**Aim**

To write a java program for simulating RARP protocols using UDP

**ALGORITHM**

**Client:**

1.Start the program

2. using datagram sockets UDP function is established.

2.Get the MAC address to be converted into IP address.

3.Send this MAC address to server.

4.Server returns the IP address to client.

**Server:**

1. Start the program.

2. Server maintains the table in which IP and corresponding MAC addresses are stored.

3. Read the MAC address which is send by the client.

4. Map the IP address with its MAC address and return the IP address to client.

**Program**

**Client:**

import java.io.\*;

import java.net.\*;

import java.util.\*;

class Clientrarp

{

public static void main(String args[])

{

try

{

DatagramSocket client=new DatagramSocket();

InetAddress addr=InetAddress.getByName("127.0.0.1");

byte[] sendbyte=new byte[1024];

byte[] receivebyte=new byte[1024];

BufferedReader in=new BufferedReader(new InputStreamReader(System.in));

System.out.println("Enter the Physical address (MAC):");

String str=in.readLine(); sendbyte=str.getBytes();

DatagramPacket sender=new DatagramPacket(sendbyte,sendbyte.length,addr,1309);

client.send(sender);

DatagramPacket receiver=new DatagramPacket(receivebyte,receivebyte.length);

client.receive(receiver);

String s=new String(receiver.getData());

System.out.println("The Logical Address is(IP): "+s.trim());

client.close();

}

catch(Exception e)

{

System.out.println(e);

}

}

}

**Server:**

import java.io.\*;

import java.net.\*;

import java.util.\*;

class Serverrarp

{

public static void main(String args[])

{

try

{

DatagramSocket server=new DatagramSocket(1309);

while(true)

{

byte[] sendbyte=new byte[1024];

byte[] receivebyte=new byte[1024];

DatagramPacket receiver=new DatagramPacket(receivebyte,receivebyte.length);

server.receive(receiver);

String str=new String(receiver.getData());

String s=str.trim();

InetAddress addr=receiver.getAddress();

int port=receiver.getPort();

String ip[]={"165.165.80.80","165.165.79.1"};

String mac[]={"6A:08:AA:C2","8A:BC:E3:FA"};

for(int i=0;i<ip.length;i++)

{

if(s.equals(mac[i]))

{

sendbyte=ip[i].getBytes();

DatagramPacket sender=new DatagramPacket(sendbyte,sendbyte.length,addr,port);

server.send(sender);

break;

}

}

break;

}

}

catch(Exception e)

{

System.out.println(e);

}

}

}

**Output:**

I:\ex>java Serverrarp12

I:\ex>java Clientrarp12

Enter the Physical address

(MAC): 6A:08:AA:C2

The Logical Address is(IP): 165.165.80.80

**Result**

Thus the RARP program using UDP was executed.

**8A.Code simulating PING command**

**Aim:**

 To Write the java program for simulating ping command

**Algorithm:**

Step 1: start the program.

Step 2: Include necessary package in java.

Step 3: To create a process object p to implement the ping command.

Step 4: declare one BufferedReader stream class object.

Step 5: Get thedetails of the server

         5.1: length of the IP address.

         5.2: time required to get the details.

         5.3: send packets , receive packets and lost packets.

         5.4: minimum ,maximum and average times.

Step 6: print the results.

Step 7:Stop the program.

**Program:**

import java.io.\*;

public class ping1

{

public static void runSystemCommand(String Command)

{

try{

Process p=Runtime.getRuntime().exec(Command);

BufferedReader InputStream=new BufferedReader(new InputStreamReader(p.getInputStream()));

String s="vvv";

while((s=InputStream.readLine())!=null)

{

System.out.println(s);

}

}

catch(Exception e)

{

e.printStackTrace();

}

}

public static void main(String[]args)

{

String Ip="localhost";

runSystemCommand("ping " +Ip);

java.util.Date date=new java.util.Date();

System.out.println(date);

}

}

**8B.TRACEROUTE COMMAND**

**Program:**

import java.io.BufferedReader;

import java.io.InputStreamReader;

public class traceroutecmd

{

     public static void runSystemCommand(String command)

     {

          try

          {

              Process p = Runtime.getRuntime().exec(command);

              BufferedReader inputStream = new BufferedReader(

                        new InputStreamReader(p.getInputStream()));

              String s = &quot;&quot;;

              while ((s = inputStream.readLine()) != null)

                   System.out.println(s);

          }

          catch (Exception e)

          {

          }

     }

     public static void main(String[] args)

     {

          // String ip = &quot;www.google.co.in&quot;;

          // String ip = &quot;127.0.0.1&quot;;

          String ip = &quot;www.cp-algorithms.com&quot;;

          runSystemCommand(&quot;tracert &quot; + ip);

     }

}

**9. RPC (Remote Procedure Call)**

**Aim:**

To write a java program to implement RPC (Remote Procedure Call)

**Algorithm:**

1. Start the program.
2. Get the frame size from the user
3. To create the frame based on the user request.
4. To send frames to server from the client side.
5. If your frames reach the server it will send ACK signal to client otherwise it will send NACK signal to client.
6. Stop the program

**Program:**

**Client:**

import java.io.\*;

import java.net.\*;

import java.util.\*;

class Clientrpe{

public static void main(String args[]){

try{

BufferedReader in=new BufferedReader(new InputStreamReader(System.in));

Socket clsct-new Socket("127.0.0.1",139);

DataInputStream din=new DataInputStream(clsct.getInputStream());

DataOutputStream dout=new DataOutputStream(clsct.getOutputStream()); System.out.println("Enter String");

String str-in.readLine();

dout.writeBytes(str+'\n');

clsct.close();

}

catch (Exception e){

System.out.println(e);

}

}

}

**Server:**

import java.io.\*;

import java.net.\*;

import java.util.\*;

class Serverrpc {

public static void main(String args[]) {

try {

ServerSocket obj=new ServerSocket(139);

while(true) {

Socket obj1=obj.accept();

DataInputStream din=new DataInputStream(obj1.getInputStream());

DataOutputStream dout=new DataOutputStream(obj1.getOutputStream());

String str=din.readLine();

Process p=Runtime.getRuntime().exec(str);

}

}

catch(Exception e) {

System.out.println(e);

}

}

}

**Output:**

Server

Y:\networks\remote>java Serverrpc

Client

Y:\networks\remote>java Clientrpc

Enter String

Calc

**10. Study of Network simulator (NS) and Simulation of Congestion Control Algorithms using NS**

**AIM**

To Study of Network simulator (NS) and Simulation of Congestion Control Algorithms using NS.

**Network Simulator**

Network Simulator (NS) is simply a discrete event-driven network simulation tool for studying the dynamic nature of communication networks. Network Simulator provides substantial support for simulation of different protocols over wired and wireless networks.

Network simulation offers an efficient, cost-effective way to assess how the network will behave under different operating conditions. Simulation results can be analyzed to assess network performance, identify potential problems, understand the root cause, and resolve the issues prior to deployment.

**Examples of network simulators**

* Ns2 (Network Simulator 2)
* Ns3 (Network Simulator 3)
* OPNET
* OMNeT++
* NetSim
* REAL
* QualNet
* J-Sim

**Congestion**

It is a state occurring in network layer when the message traffic is so heavy that it slows down network response time.

Congestion control refers to the techniques used to control or prevent congestion. Congestion control techniques can be broadly classified into two categories:

* Open Loop Congestion Control
* Closed Loop Congestion Control

**Effects** **of Congestion**

* As delay increases, performance decreases.
* If delay increases, retransmission occurs, making situation worse.

**Congestion control algorithms**

* Leaky Bucket Algorithm
* Token bucket Algorithm

**Leaky Bucket Algorithm:**

Let us consider an example to understand, Imagine a bucket with a small hole in the bottom. No matter at what rate water enters the bucket, the outflow is at constant rate. When the bucket is full with water additional water entering spills over the sides and is lost.

Similarly, each network interface contains a leaky bucket and the following steps are involved in leaky bucket algorithm:

1. When host wants to send packet, packet is thrown into the bucket.
2. The bucket leaks at a constant rate, meaning the network interface transmits packets at a constant rate.
3. Bursty traffic is converted to a uniform traffic by the leaky bucket.
4. In practice the bucket is a finite queue that outputs at a finite rate.

**Token bucket Algorithm:**

Need of token bucket Algorithm:-

The leaky bucket algorithm enforces output pattern at the average rate, no matter how bursty the traffic is. So in order to deal with the bursty traffic we need a flexible algorithm so that the data is not lost. One such algorithm is token bucket algorithm.

Steps of this algorithm can be described as follows:

1. In regular intervals tokens are thrown into the bucket. ƒ
2. The bucket has a maximum capacity. ƒ
3. If there is a ready packet, a token is removed from the bucket, and the packet is sent.
4. If there is no token in the bucket, the packet cannot be sent.

**RESULT:**

Thus the study of Network simulator (NS2) was studied.

**11.Case study on routing algorithms**

**i. Link State routing protocol**

**AIM:**

To study the link state routing algorithm.

**Link State routing:**

Link state routing is a method in which each router shares its neighborhood’s knowledge with every other router in the internetwork. In this algorithm, each router in the network understands the network topology then makes a routing table depend on this topology.

Multipath routing techniques enable the use of multiple alternative paths. In case of overlapping/equal routes, the following elements are considered in order to decide which routes get installed into the routing table (sorted by priority):

1. Prefix-Length: where longer subnet masks are preferred (independent of whether it is within a routing protocol or over different routing protocol

2. Metric: where a lower metric/cost is preferred (only valid within one and the same routing protocol)

3. Administrative distance: where a lower distance is preferred (only valid between different routing protocols)

**Advantages**

* Fast convergence: changes are reported immediately by the source affected
* Robustness against routing loo
* Router know the topology

**Disadvantages**

* Significant demands on memory and processing resources
* Requires very strict network design
* Requires a knowledgeable network administrator

**ii. Flooding**

**Algorithm:**

There are several variants of flooding algorithm. Most work roughly as follows:

1. Each node acts as both a transmitter and a receiver.

2. Each node tries to forward every message to every one of its neighbors except the source node.

This results in every message eventually being delivered to all reachable parts of the network. Algorithms may need to be more complex than this, since, in some case, precautions have to be taken to avoid wasted duplicate deliveries and infinite bops, and to allow messages to eventually expire from the system. A variant of flooding called selective flooding partially addresses these issues by only sending packets to routers in the same direction. In selective flooding the routers don't send every incoming packet on every line but only on those lines which are going approximately in the right direction.

**Advantage**

- simple and robust

- the best approach for short packet lengths, small number o participants in highly mobile networks with light traffic

**Disadvantage**

- High overhead

- Broadcasting is unreliable

lack of acknowledgements

hidden, exposed terminals lead to data loss or delay

**iii. Distance vector routing protocol**

In computer communication theory relating to packet-switched networks, a distance vector routing protocol is one of the two major classes of routing protocols, the other major class being the link-state protocol. Distance-vector routing protocol use the Bellman-Ford algorithm, Ford-Fulkerson algorithm, or DUAL FSM (in the case of Cisco Systems' protocols) to calculate paths.

A distance-vector routing protocol requires that a router informs its neighbors of topology changes periodically. Compared to link-state protocols, which require a router to inform all the nodes in a network of topology changes, distance-vector routing protocols have less computational complexity and message overhead.

Routers using distance-vector protocol do not have knowledge of the entire path to a destination. Instead they use two methods:

1. Direction in which router or exit interface a packet should be forwarded.

2. Distance from its destination

**Advantages:**

* Simplicity
* Low resource requirements
* Minimum link bandwidth

**Disadvantages:**

* Slow convergence
* Limited scalability
* Potential for routing loops (coming)

**RESULT:**

Thus the case study about the different routing algorithms to select the network path with is optimum and economical during data transfer was performed.