Expt. No. 6

Expt. Name. DAEMON PROGRAM

Page No.

Date : 30/09/2021

Aim: A daemon (produced DEE-muhn) is a program that runs continuously and exists for the purpose of handling periodic service requests that a computer system expects to receive.

Algorithm:

Step1: Initialize public class DaemonThread which extends Thread.

Step2: Another & Call the public class DaemonThread which takes constructor String name as parameter and return super(name)

Step3: Initialize run() method to check whether the thread is daemon or user

Step4: In main method create t_1, t_2, t_3 as objects of DaemonThread.

Step5: Set t_1 to daemon then start first two threads and again set t_3 to daemon and start t_3.

Step6: Stop

Expt. No.

Expt. Name

Page No.

Date

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
    }
}
```

Result: Daemon Thread prog' executed sucessfully

```
omputer Network Lab_98269ee1\\bin" DaemonThread
t1 is Daemon thread
t2 is User thread
t3 is Daemon thread
```

Expt. No. 7.

Page No.

Expt. Name.

Date : 30/09/2021

Aim: Program to implement HTTP protocol and to print URL for the client

Algorithm:

Step1: Create the URL with Http URL connections

Step2: Define the Http Protocol for Client Connections.

Step3: Get the Http connections

Step4: Print the URL for the Client

Program:

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

public class myhttp {
    public static void main(String[] args) throws
                                        IOException {
        URL url = new URL("https://www.google.com/");
        URLConnection conn = url.openConnection();
        conn.connect();
        InputStreamReader content = new
                        InputStreamReader(conn.getInputStream());

        FileWriter f = new FileWriter("http.html");
        for(int i=0; i != -1; i= content.read())
```

Expt. No. _____ Page No. _____

Expt. Name. _____ Date : _____

```
        f.write((char) i);
    }
}
```

Result: HTTP protocol and to ~~print~~ print URL for client is completed successfully.


Google

Expt No. 8

Expt Name Implement FTP using TCP

Page No.

Date :

Aim: Implement FTP using TCP

Algorithm

Client:

Step1: Create a file which has to be read by server and write the content and save it as "FTP FileTest.txt".

Step2: Import the neq. io and net packages

Step3: Initialize the socket class to communicate with server

Step4: Client specifies the file name to ~~thes~~ the server

Step5: Initialize the BufferedReader class to read the filename from the terminal

Step6: Initialize the PrintWriter class to write and send the file name to the server through the socket

Step7: Initialize BufferReader class to read the file contents from the server

Step8: Repeatedly read the data till end of file equals to empty.

Expt. No. _____ Page No. _____

Expt. Name. _____ Date : _____

Step9: Close the socket stream.

Step10: Stop

Server:

Step1: Import the req. io. and net packages

Step2: Initialize the ServerSocket class and accept the client connection.

Step3: Initialize the BufferReader class for reading the client request (i.e. file name)

Step4: Initialize the FileReader class for reading the file

Step5: Initialize BufferReader class for reading the file contents line by line.

Step6: Repeatedly read the file content line by line and send to the client through the socket s.

Step7: Stop.

Expt. No. _____ Page No. _____

Expt. Name. _____ Date : _____

Program

Server

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

public class FTPServer {
    public static void main(String[] args) throws
                                        Exception {
        ServerSocket ss = new ServerSocket(4000);
        System.out.println("Server ready for Connection");

        Socket s = ss.accept();

        System.out.println("Connection is successful & waiting for chatting");

        InputStream istream = s.getInputStream();

        BufferedReader fileread = new BufferedReader(
                                new InputStreamReader
                                (istream));

        String fname = fileRead.readLine();

        BufferedReader contentRead = new
                BufferedReader(new FileReader(fname));
```

Expt. No. _____ Page No. _____

Expt. Name. _____ Date : _____

```
OutputStream ostream
            = s.getOutputStream();

PrintWriter pwrite = new PrintWriter(
                        ostream, true);
String str;

while (( str = contentRead.readLine()) != null)
    pwrite.println(str);

s.close();
ss.close();
pwrite.close();
fileRead.close();
contentRead.close();
    }
}
```

Result FTP was successfully implemented using TCP.

FTPClient.java > FTPClient

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

public class FTPClient {
    Run | Debug
    public static void main(String args[]) throws Exception {
        Socket s = new Socket(InetAddress.getLocalHost(), 4000);
        // reading the file name from keyboard. Uses input stream
        System.out.println("Enter the file name");
        BufferedReader keyRead = new BufferedReader(new InputStreamReader(System.in));
        String fname = keyRead.readLine();
        // sending the file name to server. Uses PrintWriter
        OutputStream ostream = s.getOutputStream();
        PrintWriter pwrite = new PrintWriter(ostream, true);
        pwrite.println(fname);
        // receiving the contents from server. Uses input stream
        InputStream istream = s.getInputStream();
        BufferedReader socketRead = new BufferedReader(new InputStreamReader(istream));
        String str;
        while ((str = socketRead.readLine()) != null) // reading line-by-line
        {
            System.out.println(str);
        }
        pwrite.close();
        socketRead.close();
        keyRead.close();
    }
}
```

PROBLEMS 8 OUTPUT DEBUG CONSOLE TERMINAL

bphar@LAPTOP MINGW64 /d/AAdityAA/SIST/Sem 5/Computer Network Lab (master)

bphar@LAPTOP MINGW64

Client

```
$ java FTPClient
Enter the file name
FTPFileTest.txt
Hey There FTP program worked successfully!!
```

Server

```
$ java FTPServer
Server ready for connection
Connection is successful and wating for chatting
```

Expt. No. 9 Page No.

Expt. Name. Traceroute Command Date :

Aim: To implement the traceroute command program

Algorithm:

Step1: Import the req. packages

Step2: Create public class traceroute command

Step3: start main function

Step4: Initialize the process 'p' runtime

Step5: the string s

Step6: While loop 's' is ~~equal~~ Input Stream read

Step7: Print s

Step8: str using catch exception 'e'

Step9: string ip input

Step10 stop.

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 = "";

            while((s = inputStream.readLine()) != null)
                System.out.println(s);
        } catch (Exception e)
        {
        }
    }
    public static void main(String args[])
    {
        String ip = "www.google.com";

        runSystemCommand("tracert " + ip);
    }
}
```

```
Tracing route to www.google.co.in [172.217.31.195]
over a maximum of 30 hops:

  1     4 ms     1 ms     1 ms  192.168.0.1
  2     4 ms   385 ms     6 ms  10.215.12.1
  3     7 ms     5 ms     5 ms  103.81.238.249
  4     7 ms     7 ms     8 ms  103.81.239.38
  5    47 ms    11 ms    10 ms  108.170.253.97
  6     9 ms     8 ms    10 ms  74.125.253.17
  7    10 ms     8 ms     7 ms  maa03s28-in-f3.1e100.net [172.217.31.195]

Trace complete.
```

Expt. No. 10

Expt. Name. PING COMMAND

Page No.

Date :

Aim: To execute Ping command program

Algorithm

Step1: Import req. packages

Step2: Start the Process 'p' with runtime to execute "Command"

Step3: Create an instance of BufferedReader

Step4: Initialize String s to "vvv"

Step5: Start a while loop with cond^n s!=null

Step6: print(s)

Step7: Catch the exception e

Step8: In main init ip as "localhost"

Step9: Call the func^n with String "ping"+ip as parameter

Step10: print the Date object date.

Expt. No. _____ Page No. _____

Expt. Name. _____ Date : _____

Program

```
import java.io.*;

import

public class ping {
    public class 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(e);
    }
    public static void main (String args[])
    {
            String ip = "localhost";

            runSystemCommand ("ping "+ip);
```

Expt. No. _____ Page No. _____

Expt. Name. _____ Date : _____

```
        java.util.Date date = new Date();
        System.out.println(date);
    }
}
```

Result: The Ping Command prog. is executed successfully.

```
Pinging LAPTOP [::1] with 32 bytes of data:
Reply from ::1: time<1ms
Reply from ::1: time<1ms
Reply from ::1: time<1ms
Reply from ::1: time<1ms

Ping statistics for ::1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
Wed Oct 13 14:34:50 IST 2021
```

Expt. No. 1

Expt. Name. Case Study

Page No.

Date :

(a) Distance Vector Routing

-> A DVR Protocol requires that a router inform it's neighbours of topology changes periodically.

->Also known as the old ARPANET routing algorithm (or Bellman-Ford algorithm)

Algorithm

Step1: A router transmits it's distance vector to each of its neighbours in a routing packet

Step2: Each router receives and saves the most recently received distance vector from each of it's neighbours.

Step3: A router it's distance ~~vect~~ vector when:

(a) It recieves a DV from a neighbour containing different information than before

(b) It discovers that a link to a neighbour has gone down

Advantages:

-> It is simpler to configure and maintain than link state routing.

Expt. No. _____ Page No. _____

Expt. Name. _____ Date : _____

Disadvantages:

→ Slower than link state

→ is at risk from the count-to-infinity problem

Link State - Routing Protocol:-

→ Link State routing are one of the main classes of routing protocols used in packet switching networks

→

Advantages

1. Fast Network convergence
2. Topological Maps
3. Hierarchical Design
4. Event-driven Updates

Disadvantages:

1. Memory requirements
2. Processing requirements
3. Bandwidth Requirements