CS553 Cloud Computing
Programming Assignment 1
Source Code

Submitted by:
Chiranjeevi Ankamredy
A20359837

1.CPU.

A.CPU.JAVA

```
 import java.util.Scanner;

class  NumberOfExperiments extends Thread // This class is for number of experiments in cpu
{
  public static int NumThreads;


         public NumberOfExperiments(int i) {  // This method is for number of experiments in
cpu

                  NumThreads=i;
         // TODO Auto-generated constructor stub
}

         public void run()
         {


                           long FlopsTotalTime = FlopsTime(); //Flops time calculation
                           long IopsTottslTime = IopsTime();   //Iops time calculation
         float NumGflops = calculationFlops(FlopsTotalTime); ////Flops time calculation
         float NumGiops = calculationFlops(IopsTottslTime); // //number of Giops calculation


                           System.out.println("  Thread-"+NumThreads+"  Time for FLOPS    :"+
FlopsTotalTime+"ms");
                           System.out.println("  Thread-"+NumThreads+"  Time for IOPS      :"+
IopsTottslTime+"ms");
                       System.out.println("  Thread-"+NumThreads+"  Number of GFLOPS  :"+
NumGflops*23*3);
                           System.out.println("  Thread-"+NumThreads+"  Number of GIOPS
         :"+ NumGiops*24);
```

```
    }

    public static float calculationFlops(float value){   //Flops time calculation
            value = value/1000;
            float flops = ((1000000000)/value)/1000000000;
            return flops;
    }


    private long IopsTime() {  //Iops time calculation
            // TODO Auto-generated method stub
            long k=9,m=12,h=2;
    long i;
       long sum=0;
      long Initime = System.currentTimeMillis();
       for(i=1;i<=1000000000;i++)
       {
            sum=sum*k+5*i+i*i-m*i-h;

       }
      long FinalTime = System.currentTimeMillis();
long OperationIntTime=FinalTime-Initime;

    return OperationIntTime;
    }


    private long FlopsTime() {   //Flops time calculation
            // TODO Auto-generated method stub
             double k=6.5f;
      double m=3.2f;
             double sum=1f;
                   long i;
             long Initime1 = System.currentTimeMillis();
             for(i=1;i<=1000000000;i++)
               {
                    sum=sum*k+m/k*k+1/m*k*5*m*k*m*k+m+m-m;
               }
             long FinalTime1 = System.currentTimeMillis() ;
       long OperationFloatTime=FinalTime1-Initime1;
```

```
                return OperationFloatTime;
        }

}


class Cpu
{


        public static void main(String args[]) throws InterruptedException
     {


                                 int NumThreads;
          Scanner scan = new Scanner(System.in);

          NumberOfExperiments[] Expment = new NumberOfExperiments[8];  //Number of
Experiments



                   char ch;
                                do{

         System.out.println("1.one thread ");
         System.out.println("2.Two threads");
           System.out.println("3.Four Threads ");
           System.out.println("Enter How many threads to Run: \n");
         int choice = scan.nextInt();



                 switch (choice)
                 {
                  case 1 :  for(int i=1; i<=1; i++)
                             {
                               //NumberOfExperiments.NumThreads = i;
```

```
                    Expment[i-1] = new NumberOfExperiments(i);
                    Expment[i-1].start();

                    System.out.println(" Thread "+i+" Started");
                       Thread.sleep(1000);
                  }

       break;


 case 2 :    for(int i=1; i<=2; i++)
                     {

                               Expment[i-1] = new NumberOfExperiments(i);
                               Expment[i-1].start();
                                System.out.println(" Thread "+i+" Started");
                                Thread.sleep(1000);
                     }

         break;

 case 3 :for(int i=1; i<=4; i++)
                      {

                               Expment[i-1] = new NumberOfExperiments(i);
                               Expment[i-1].start();
                                System.out.println(" Thread "+i+" Started");
                                Thread.sleep(1000);
                      }

       break;

    }

    System.out.println("Do you want to continue (Type y or n) \n");
ch = scan.next().charAt(0);
 String str = Character.toString(ch);

             }while (ch == 'Y'|| ch == 'y');
    }




 }
```

B.EXTRA EXPERIMENT

a.Cpuflops.java

```
               import java.util.Scanner;

class  NumberOfExperiments extends Thread  //This class is for
600 samples calculation for Flops
{
   public static int NumThreads;
    public static int b;


     public NumberOfExperiments(int d, int z) {

          NumThreads=d;

                b=z;

}

     public void run()
     {


               long FlopsTotalTime = FlopsTime();
               float NumGflops =
calculationFlops(FlopsTotalTime);

               int q,y=1;
                        q=b%60;
               if(q==0)
                        {
                          System.out.println(" ------------------
------------ Time "+y+" miniute-----------------");
               System.out.println("  Thread-"+NumThreads+"  Time
for FLOPS :"+ FlopsTotalTime+"ms");
                        System.out.println("  Thread-
"+NumThreads+"  Number of GFLOPS   :"+ NumGflops*23*3);

                         y++;

               }



     }

     public static float calculationFlops(float value){ //Flops
calculation

          float flops = ((10000000)/value)/100000;
          return flops;
     }





     private long FlopsTime() { //Flops time

           double k=6.5f;
                 double m=3.2f;
            double sum=1f;
               long i;
            long Initime1 = System.currentTimeMillis(); //
current Sytem running in milliseconds

            for(i=1;i<=10000000;i++)
               {
                sum=sum*k+m/k*k+1/m*m*m*m*m*m*m*m-k-k-k-k-k-k;

              }
           long FinalTime1 = System.currentTimeMillis() ;
          long OperationFloatTime=FinalTime1-Initime1;

                 //long time=OperationFloatTime/1000;
          return OperationFloatTime;
     }

}


class Cpu1
{


     public static void main(String args[]) throws
InterruptedException
        {



                     int NumThreads;
                 Scanner scan = new Scanner(System.in);
           NumberOfExperiments[] Expment = new
NumberOfExperiments[8];                   //No.of Experiments


                        for(int z=1; z<=600; z++){

                       for(int d=1; d<=4; d++)
                                     {

                                        Expment[d-1] = new
NumberOfExperiments(d,z);
                                        Expment[d-1].start();

                                         //Thread.sleep(1000);
                                     }



                     }




        }
```

```
   }
```

b.CpuIops.java

```
            import java.util.Scanner;

class  NumberOfExperiments extends Thread  //This class is for
600 samples calculation for iops

{
   public static int NumThreads;
    public static int b;


     public NumberOfExperiments(int d, int z) {

          NumThreads=d;

                b=z;

}

     public void run()
     {


               long IopsTottslTime = IopsTime(); //Iops TIME

               float NumGiops = calculationFlops(IopsTottslTime); //Total calculation of
flops

               int q,y=1;
                        q=b%60;
               if(q==0)
                        {
                          System.out.println(" ------------------
------------ Time "+y+" miniute-----------------");
               System.out.println("  Thread-"+NumThreads+"  Time
for FLOPS :"+ FlopsTotalTime+"ms");
                        System.out.println("  Thread-
"+NumThreads+"  Number of GFLOPS   :"+ NumGiops*23*3);
```

```
                         y++;

               }



     }

     public static float calculationFlops(float value){   //Total
calculation of flops


          float flops = ((10000000)/value)/100000;
          return flops;
     }





     private long IopsTime() {  //Total calculation of iops

          // TODO Auto-generated method stub
          long k=9,m=12,h=2;
     long i;
       long sum=0;
      long Initime = System.currentTimeMillis();
       for(i=1;i<=1000000000;i++)
       {
            sum=sum*k+5*i+i*i-m*i-h;

       }
      long FinalTime = System.currentTimeMillis();
  long OperationIntTime=FinalTime-Initime;

     return OperationIntTime;
     }

}


class Cpu1
{
     public static void main(String args[]) throws
InterruptedException
        {


                     int NumThreads;
                 Scanner scan = new Scanner(System.in);

                 NumberOfExperiments[] Expment = new
NumberOfExperiments[8];  //Number of Experiments


                        for(int z=1; z<=600; z++){

                       for(int d=1; d<=4; d++)
                                     {

                                        Expment[d-1] = new
NumberOfExperiments(d,z);
                                        Expment[d-1].start();

                                         //Thread.sleep(1000);
                                     }



                     }




        }





  }
```

2.DISK

A.DiskSeq.java

```
     import java.io.BufferedWriter;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileWriter;
import java.io.RandomAccessFile;
import java.util.Scanner;

class  BlocksizeExec extends Thread   //Disk Sequence Access for Each Block Size
{
   public static long BufferLen;
   public static float[] AverageWrite = new float[2];
        public static double[] AverageRead = new double[2];
        public static float[] AverageLatency = new float[2];
        public static float[] WriteTime = new float[2];
        public static double[] ReadTime = new double[2];
        public static float[] LatencyTime = new float[2];

        public static int ThreadNum;

        //public BlocksizeExec(int k)
        //{

//              BufferLen=k;

 //  }



        public static void Exec() throws InterruptedException  //Disk Sequence Access
Excecution for Each Block Size
        {
                BlocksizeExec[] run = new BlocksizeExec[8];
                for (int j = 0; j < 2; j++)
                {
                        int NumOfThreads = (int)Math.pow(2, j); //No.of threads Running
                        for (int i = 0; i < NumOfThreads; i++)
                        {
                                run[i] = new BlocksizeExec();
                                run[i].start();
                                run[i].ThreadNum = j;
                                Thread.sleep(100);
                        }
```

```
            BlocksizeExec.AverageWrite[j] = ((WriteTime[j] / NumOfThreads)/
1000000000)*(1048576/BufferLen);
            BlocksizeExec.AverageRead[j] = ((ReadTime[j] / NumOfThreads)/
1000000000)*(1048576/BufferLen);
            BlocksizeExec.AverageLatency[j] = (LatencyTime[j] / NumOfThreads) /
1000000;
        }
    }
    public void run()
    {

        try
        {
            FileWrite(BlocksizeExec.BufferLen);     //Calculating throughput
            FileRead();         //Caluclating Read time
            Latency();         //Calculating Latency
        }
        catch (Exception e) {
        e.printStackTrace();
        }
    }



    public static void Latency()    //Calculating Latency
    {  try{
        File file = new File("/home/chiru/Desktop/chiru.txt");
        RandomAccessFile access = new RandomAccessFile(file, "rw");
        long StartTime = System.nanoTime();

        access.seek(file.length());
         long FinalTime = System.nanoTime();
      long TotalTime=FinalTime-StartTime;
      long latency = TotalTime ;
        access.close();
          BlocksizeExec.LatencyTime[ThreadNum] =
BlocksizeExec.LatencyTime[ThreadNum]
                + (latency/BufferLen);
    }
    catch (Exception e) {
        e.printStackTrace();
        }


        }



public static void FileWrite(long BufferLen) throws Exception    //Calculating throughput
{

          try{
                  StringBuffer buffer = new StringBuffer();   //Buffer Storage for Reafd or Write
Data
                        for (int i = 0; i < BufferLen; i++)
                        {
                                buffer.append("2");
                        }
                        File file = new File("/home/chiru/Desktop/chiru.txt");  //File creation
                        BufferedWriter bw = new BufferedWriter(new FileWriter(
                                        file.getAbsoluteFile()));
                         long StartTime = System.nanoTime();
                        bw.write(buffer.toString());
                         long FinalTime = System.nanoTime();
                     long TotalTime=FinalTime-StartTime;   //Total time for reading
                 BlocksizeExec.WriteTime[ThreadNum] =
BlocksizeExec.WriteTime[ThreadNum]+ TotalTime;
                        bw.close();

      }
  catch (Exception e) {
        e.printStackTrace();
}
        }

public static void FileRead() throws Exception {      ////Calculating File Reading Time

        try{
                File file = new File("/home/chiru/Desktop/chiru.txt");;
                FileInputStream fis = new FileInputStream(file);
                byte[] data = new byte[(int) file.length()];

                long StartTime = System.nanoTime();
                fis.read(data);
                long FinalTime = System.nanoTime();
             long TotalTime=FinalTime-StartTime;
             BlocksizeExec.ReadTime[ThreadNum] = BlocksizeExec.ReadTime[ThreadNum]+
TotalTime;

                fis.close();
            }
        catch (Exception e) {
        e.printStackTrace();
        }
}



        public static void result() {              //Printing Result
                for (int i = 0; i < 2; i++) {
                        System.out.println("Thread-" +(int)Math.pow(2, i) + " Write Speed  in
Mbps   : " + (1 / BlocksizeExec.AverageWrite[i]));
                        System.out.println("Thread-" +(int)Math.pow(2, i) + " Read  Speed  in
Mbps   : " + (1 / BlocksizeExec.AverageRead[i]));
                        System.out.println("Thread-" +(int)Math.pow(2, i) + " Latency Time in
mSec   : " + BlocksizeExec.AverageLatency[i] + "\n");
                }
        }


}

class DiskSeq    //Disk Sequence Access for Each Block Size
{



        public static void main(String args[]) throws InterruptedException
    {
                    int Blocksize;
                    Scanner scan = new Scanner(System.in);
```

```
          System.out.println("1.1B ");
         System.out.println("2.1KB");
          System.out.println("3.1MB ");
          System.out.println("Select the BockSize: \n");
         int choice = scan.nextInt();


                switch (choice)
                {
                 case 1 :
                                 Blocksize = 1;
                                 BlocksizeExec.BufferLen = Blocksize;
                                      BlocksizeExec.Exec();
                                      System.out.println("Sequential R/W Speeds for Block Size:
" + Blocksize + " Byte \n");
                                      BlocksizeExec.result();
                                      break;


             case 2 :    Blocksize = 1024;
                    BlocksizeExec.BufferLen = Blocksize;
                                              BlocksizeExec.Exec();
                                              System.out.println("Sequential R/W Speeds for
Block Size        : " + Blocksize + " Bytes \n");
                                              BlocksizeExec.result();
                                              break;

             case 3 :    Blocksize = 1024*1024;
                    BlocksizeExec.BufferLen = Blocksize;
                                              BlocksizeExec.Exec();
                                              System.out.println("Sequential R/W Speeds for
Block Size        : " + Blocksize + " Bytes \n");
                                              BlocksizeExec.result();
                                              break;



                }
```

```
            }


}
```

 B.DiskRandom.java

```
 import java.io.File;
import java.io.RandomAccessFile;
import java.util.Scanner;

class  BlocksizeExec extends Thread  //Disk Random Access for Each Block Size
{
  public static long BufferLen;
  public static float[] AverageWrite = new float[2];
    public static double[] AverageRead = new double[2];
    public static float[] AverageLatency = new float[2];
    public static float[] WriteTime = new float[2];
    public static double[] ReadTime = new double[2];
    public static float[] LatencyTime = new float[2];

    public static int ThreadNum;

    //public BlocksizeExec(int k)
    //{

//          BufferLen=k;

 //  }



    public static void Exec() throws InterruptedException   //Execution for Each blocksize
    {
        BlocksizeExec[] run = new BlocksizeExec[8];
        for (int j = 0; j < 2; j++)
        {
            int NumOfThreads = (int)Math.pow(2, j);
            for (int i = 0; i <NumOfThreads; i++)  //Threads Rummning in a loop
            {
                run[i] = new BlocksizeExec();
                run[i].start();
                run[i].ThreadNum = j;

                Thread.sleep(100);
            }


            BlocksizeExec.AverageWrite[j] = ((WriteTime[j] / NumOfThreads)/
1000000000)*(1048576/BufferLen);
            BlocksizeExec.AverageRead[j] = ((ReadTime[j] / NumOfThreads)/
1000000000)*(1048576/BufferLen);

            BlocksizeExec.AverageLatency[j] = (LatencyTime[j] / NumOfThreads) /
1000000;
        }
    }
    public void run()
    {

        try
        {
            FileWrite(BlocksizeExec.BufferLen);  //Calculating throughput
            FileRead();  //Calculatig file reading
            Latency();    //Calculating Latency
        }
        catch (Exception e) {
        e.printStackTrace();
        }
    }




    public static void Latency()    //Calculating Latency
    {  try{
        File file = new File("/home/chiru/Desktop/chiru.txt");
  RandomAccessFile File1 = new RandomAccessFile(file, "rw");
  long StartTime = System.nanoTime();
```

```
    for(long i=0;i<BufferLen;i++)
         {
      File1.seek((file.length()+i)-BufferLen);
         }
           long FinalTime = System.nanoTime();
    long TotalTime=FinalTime-StartTime;
    long latency = TotalTime ;

    File1.close();
    BlocksizeExec.LatencyTime[ThreadNum] = BlocksizeExec.LatencyTime[ThreadNum]
                          + (latency/BufferLen);
      }
      catch (Exception e) {
      e.printStackTrace();
      }


      }



public static void FileWrite(long BufferLen) throws Exception    //Calculating throughput
{

        try{
             File file = new File("/home/chiru/Desktop/chiru.txt");  //File creation
    RandomAccessFile File1 = new RandomAccessFile(file, "rw");

    File1.seek(file.length());

    if(BufferLen==1)
    {
      for(long i=0;i<100;i++)
         {
      File1.write('2');
         }
    }
    else
    {
      for(int i=0;i<BufferLen;i++)
```

```
         {
       File1.write('2');
         }
    }
    long StartTime = System.nanoTime();
    for(long i=0;i<BufferLen*10;i++)
         {
       File1.seek((file.length()+i)-BufferLen);
       File1.write('1');
         }
    long FinalTime = System.nanoTime();
    long TotalTime=FinalTime-StartTime;
    BlocksizeExec.WriteTime[ThreadNum] = BlocksizeExec.WriteTime[ThreadNum]+
TotalTime;
    File1.close();
    }
  catch (Exception e) {
       e.printStackTrace();
}
       }

public static void FileRead() throws Exception {       //Calculating File read tIME

       try{
       File file = new File("/home/chiru/Desktop/chiru.txt"); //File creation
  RandomAccessFile File1 = new RandomAccessFile(file, "r");
  File1.seek(file.length());

  byte[] data = new byte[1];

  long StartTime = System.nanoTime();
  for(long i=0;i<BufferLen;i++)
  {
       File1.seek((file.length()+i)-BufferLen);
       File1.read(data);
  }
  long FinalTime = System.nanoTime();

  long TotalTime=FinalTime-StartTime;          //Total TIME Calculation
  BlocksizeExec.ReadTime[ThreadNum] = BlocksizeExec.ReadTime[ThreadNum]+ TotalTime;

    System.out.println(" hi " + BlocksizeExec.ReadTime[ThreadNum]);
  File1.close();

        }
        catch (Exception e) {
        e.printStackTrace();
        }
}



        public static void result() {          //Prinitng Result
                for (int i = 0; i < 2; i++) {

                        System.out.println("Thread-" +(int)Math.pow(2, i) + " Write Speed  in
Mbps   : " + (1 / BlocksizeExec.AverageWrite[i]));
                        System.out.println("Thread-" +(int)Math.pow(2, i) + " Read  Speed  in
Mbps   : " + (1 / BlocksizeExec.AverageRead[i]));
                        System.out.println("Thread-" +(int)Math.pow(2, i) + " Latency Time in
mSec   : " + BlocksizeExec.AverageLatency[i] + "\n");
                }
        }


}

class DiskRandom   //DiskRandom Main classs
{


        public static void main(String args[]) throws InterruptedException
     {
                      long Blocksize;
                      Scanner scan = new Scanner(System.in);
          System.out.println("1.1B ");
        System.out.println("2.1KB");
          System.out.println("3.1MB ");
          System.out.println("Select the BockSize: \n");
          int choice = scan.nextInt();


                 switch (choice)
                 {
                  case 1 :
                                    Blocksize = 1;
                                    BlocksizeExec.BufferLen = Blocksize;
                                          BlocksizeExec.Exec();
                                          System.out.println("Random R/W Speeds for Block Size    :
" + Blocksize + " Byte \n");
                                          BlocksizeExec.result();
                                          break;



              case 2 :    Blocksize = 1024;
                      BlocksizeExec.BufferLen = Blocksize;
                                                   BlocksizeExec.Exec();
                                                   System.out.println("Random R/W Speeds for Block
Size      : " + Blocksize + " Bytes \n");
                                                   BlocksizeExec.result();
                                                   break;

              case 3 :    Blocksize = 1024*8;
                      BlocksizeExec.BufferLen = Blocksize;
                                                   BlocksizeExec.Exec();
                                                   System.out.println("Random R/W Speeds for Block
Size      : " + Blocksize + " Bytes \n");
                                                   BlocksizeExec.result();
                                                   break;




                 }




                         }
```

```
}
```

3.Network

A.TCP

a.PeerClient.java

```
 import java.net.Socket;
import java.util.Scanner;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.io.OutputStream;


public  class PeerClient implements Runnable {  //Peer Client for Upload file from Server


	@Override
	public void run() {
		try {
			UploadSpeed(file,ThreadNumber);  //UPloading file
			Thread.sleep(2000);
			DownloadSpeed(ThreadNumber);    //Downloading File
		} catch (Exception e) {
			e.printStackTrace();
		}
	}


	private int ThreadNumber;
	static File file;
	static Socket sock;
	static long buffsize;
	static long FileSize;

	public static void main(String args[])
  {

     try {
        Scanner scan = new Scanner(System.in);
   System.out.println("Enter Number of  Threads:");

            int TotalThreads = scan.nextInt();
            System.out.println("Enter Buffer Size: 1,1024,65536");
            long buffsize = scan.nextLong();


            System.out.println("Client connecting...");

            Socket sock = new Socket("localhost", 8888);  //Socket Creation

            System.out.println(" connected  successfully:)");
             file = new File("file1.txt");   //File Creation
            long FileSize = file.length();  //Calculatig File Length
            System.out.println("Size of the file              :          " + FileSize + " Byte(s) \n");

            int i;
            for ( i = 1; i <= TotalThreads; i++) {
                        PeerClient client = new PeerClient();
               client.ThreadNumber = i;
               Thread t=new Thread(client);
                        t.start();
            }


  }
 catch (Exception e) {
            }
}


    private void DownloadSpeed(int k) {
            try {
                        InputStream is = sock.getInputStream();                //Sending Stream
                        File output = new File("f1.txt");
                        FileOutputStream fos = new FileOutputStream(output);
                        int filelen;
                        byte[] data = new byte[1 * 1024];
                        System.out.println(" Thread"+k);
                        System.out.println(" File downloading started");
                        long Initime = System.currentTimeMillis();
                        while ((filelen = is.read(data)) > 0) {
                                fos.write(data, 0, filelen);
                                fos.flush();
                        }
                        fos.close();
                        is.close();
                        sock.close();
                        long FinalTime = System.currentTimeMillis();
                   long TotalTime=FinalTime-Initime;
                   float FileSizeinMb = (float)FileSize/1048576;
                        float Totalseconds = (float)TotalTime/1000;
                   float Speed = FileSizeinMb/Totalseconds ;
                                                                                //PrintiNG Network Speed

                        System.out.println(" file Dowanloaded");
                        System.out.println(" Time taken for download       :          " + TotalTime+
" MilliSecs");
                        System.out.println(" Avg download speed of         :          " + Speed + "
Mbps");
                        } catch (Exception e) {
                                e.printStackTrace();
                        }

        }

        public static void UploadSpeed(File file1, int n) {  //Uploading File
                try {
                         OutputStream os = sock.getOutputStream();

                        FileInputStream fis = new FileInputStream(file1);
                        byte[] data = new byte[(int) buffsize];
                        int FileLength;
                        System.out.println("Thread"+n);
                        System.out.println("File uploading started");

                         long Initime = System.currentTimeMillis();
                        while ((FileLength = fis.read(data)) > 0) {
                                os.write(data, 0, FileLength);
```

```
                              os.flush();
                         }
                         fis.close();
                         long FinalTime = System.currentTimeMillis();
                    long TotalTime=FinalTime-Initime;
                    float FileSizeinMb = (float)FileSize/1048576;
                         float Totalseconds = (float)TotalTime/1000;
                    float Speed = FileSizeinMb/Totalseconds ;

                         System.out.println(" File uploaded");
                         System.out.println(" Time taken for upload:        " + TotalTime + "
MilliSecs");
                         System.out.println(" Avg upload speed of    :        " + Speed + " Mbps");
                         }
                  catch (Exception e) {
                              e.printStackTrace();
                         }

          }
}
```

b.PeerServer.java

```
   import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.ServerSocket;
import java.net.Socket;

public class PeerServer implements Runnable {  //Server program

       private static ServerSocket server;
       private Socket sock; //Socket Creation

      public static void main(String[] args) {
    try {
      System.out.println("Server connecting...");
      server = new ServerSocket(1111);   //Server Port Accepting
```

```
  while (true) {
    Socket connection = server.accept(); //Peerclient request Accepted

    if (connection != null) {
         PeerServer s = new PeerServer();
         s.sock=connection;
     Thread t=new Thread(s);
                t.start();
    }
  }
}
catch (Exception e) {
  e.printStackTrace();
}
   }



   @Override
   public void run() {

         System.out.println(" connection establised successfully \n");
try {  //Streams uploading and Downloading
  InputStream is = sock.getInputStream();
  OutputStream os = sock.getOutputStream();
  File output = new File("file.txt");
  FileOutputStream fos = new FileOutputStream(output);
  byte[] data = new byte[1 * 1024];
  int filelen = 0;
                System.out.println("Initiating Server to Client communication");

                long Initime = System.currentTimeMillis();

  try {
    while ((filelen = is.read(data)) > 0) {
      fos.write(data, 0, filelen);
      fos.flush();
    }
    fos.close();
  } catch (Exception e) {
  }
```

```
        long FinalTime = System.currentTimeMillis();
                     long TotalTime=FinalTime-Initime;
        System.out.println("Network I / O Transfer Time     :           " + TotalTime+ " ms");
        System.out.println("file uploading completed! \n");

           Thread.sleep(1000);

        System.out.println("Initiating Client to Server communication");
        Initime = System.currentTimeMillis();
        FileInputStream fis = new FileInputStream(output);
        filelen = 0;
        while ((filelen = fis.read(data)) > 0) {
           os.write(data, 0, filelen);
           os.flush();
        }
        fis.close();
        os.close();
        sock.close();

         FinalTime = System.currentTimeMillis();  /Time calculating for Download file From
Server
                      TotalTime=FinalTime-Initime;

        System.out.println("Network I / O Transfer Time     :           " +TotalTime+ " ms");
                           System.out.println("File downloaded");
     }
     catch (Exception e) {
        e.printStackTrace();
     }


          }

}
```

  B.UDP

     a.UClient.java

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

```
class Files implements Serializable   //File Description Variables
{
   private static final long serialVersionUID = 1L;

   private String destinationDirectory;
   private String sourceDirectory;
   private String filename;
   private long fileSize;
   private byte[] fileData;
   private String status;

   public String getDestinationDirectory() {
      return destinationDirectory;
   }

   public void setDestinationDirectory(String destinationDirectory) {
      this.destinationDirectory = destinationDirectory;
   }

   public String getSourceDirectory() {
      return sourceDirectory;
   }

   public void setSourceDirectory(String sourceDirectory) {
      this.sourceDirectory = sourceDirectory;
   }

   public String getFilename() {
      return filename;
   }

   public void setFilename(String filename) {
      this.filename = filename;
   }

   public long getFileSize() {
      return fileSize;
   }
   public void setFileSize(long fileSize) {
      this.fileSize = fileSize;
   }

   public String getStatus() {
      return status;
   }

   public void setStatus(String status) {
      this.status = status;
   }

   public byte[] getFileData() {
      return fileData;
   }

   public void setFileData(byte[] fileData) {
      this.fileData = fileData;
   }

}



class UClient           //UDP Client Program
{

            private static String sourceFilePath = "file.txt";
            private static String destinationPath = "Udp/";

            public static Files getFileEvent() {  //File Events
               Files fileEvent = new Files();
               String fileName = sourceFilePath.substring(sourceFilePath.lastIndexOf("/") + 1,
sourceFilePath.length());
               String path = sourceFilePath.substring(0, sourceFilePath.lastIndexOf("/") + 1);
               fileEvent.setDestinationDirectory(destinationPath);
               fileEvent.setFilename(fileName);
               fileEvent.setSourceDirectory(sourceFilePath);
               File file = new File(sourceFilePath);
               if (file.isFile()) {
```

```
             try {
               DataInputStream diStream = new DataInputStream(new FileInputStream(file));
//Input Stream
               long len = (int) file.length();
               byte[] fileBytes = new byte[(int) len];
               int read = 0;
               int numRead = 0;
               while (read < fileBytes.length && (numRead = diStream.read(fileBytes, read,
                   fileBytes.length - read)) >= 0) {
                 read = read + numRead;
               }
               fileEvent.setFileSize(len);
               fileEvent.setFileData(fileBytes);
               fileEvent.setStatus("Success");
             } catch (Exception e) {
               e.printStackTrace();
               fileEvent.setStatus("Error");
             }
           } else {
             System.out.println("path specified is not pointing to a file");
             fileEvent.setStatus("Error");
           }
           return fileEvent;
         }



       public static void main(String args[]) throws Exception
   {



  try {
       System.out.println("Client  initiating...");
       DatagramSocket clientSocket = new DatagramSocket();  //DataGram Socket Creation

  String hostName;
       InetAddress IPAddress = InetAddress.getByName("localhost"); //Getting InetAddress

  byte[] incomingData = new byte[1024];  //Crating Buffer Size
```

```
   Files event = getFileEvent();

   ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
   ObjectOutputStream os = new ObjectOutputStream(outputStream);
   os.writeObject(event);

   byte[] data = outputStream.toByteArray();
   DatagramPacket sendPacket = new DatagramPacket(data, data.length, IPAddress, 6666);

          System.out.println("Size of the file : 10000 Bytes \n");

   long StartTime = System.nanoTime();
   clientSocket.send(sendPacket);
   long EndTime = System.nanoTime();
   long TotalTime = EndTime - StartTime;  //Caluculating Total Time

   System.out.println("File sent from client");
   DatagramPacket incomingPacket = new DatagramPacket(incomingData,
incomingData.length);   //Reeciving Data Packet
   clientSocket.receive(incomingPacket);//Receiving
   String response = new String(incomingPacket.getData());

   System.out.println("Response from server:" + response.trim());
   System.out.println("Time Take for 1 Thread upload        : "+TotalTime+" nanosec");
          float speed = 10000000/TotalTime;
          System.out.println("Bandwidth for 1 Thread upload          : "+speed+" Mbps");

          clientSocket.close();
   Thread.sleep(2000);
   System.exit(0);

     }
   catch (UnknownHostException e) {
   e.printStackTrace();}



   }
}
```

b.UServer.java   //ServerSide Program

```
import java.io.*;
import java.net.*;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.SocketException;

class Files implements Serializable   //File Descriptions for A file
{
   private static final long serialVersionUID = 1L;

   private static String destinationDirectory;
   private String sourceDirectory;
   private static String filename;
   private long fileSize;
   private static byte[] fileData;
   private static String status;

   public  String getDestinationDirectory() {
      return destinationDirectory;
   }

   public void setDestinationDirectory(String destinationDirectory) {
      this.destinationDirectory = destinationDirectory;
   }

   public String getSourceDirectory() {
      return sourceDirectory;
   }

   public void setSourceDirectory(String sourceDirectory) {
      this.sourceDirectory = sourceDirectory;
   }

   public  String getFilename() {
      return filename;
   }
```

```
    public void setFilename(String filename) {
        this.filename = filename;
    }

    public long getFileSize() {
        return fileSize;
    }

    public void setFileSize(long fileSize) {
        this.fileSize = fileSize;
    }

    public  String getStatus() {
        return status;
    }

    public void setStatus(String status) {
        this.status = status;
    }

    public  byte[] getFileData() {
        return fileData;
    }

    public void setFileData(byte[] fileData) {
        this.fileData = fileData;
    }

}

public class UServer {   //udp Server main Class



             private static Files Files=null;

    public static void createAndWriteFile() //Crate a file and Writing File
    {
        String outputFile = Files.getDestinationDirectory() + Files.getFilename();
        if (!new File(Files.getDestinationDirectory()).exists()) {
```

```
      new File(Files.getDestinationDirectory()).mkdirs();
    }
    File dstFile = new File(outputFile);
    FileOutputStream fileOutputStream = null;
    try {
      fileOutputStream = new FileOutputStream(dstFile);
      fileOutputStream.write(Files.getFileData());
      fileOutputStream.flush();
      fileOutputStream.close();
      System.out.println("Output file : " + outputFile + " is successfully saved ");

    } catch (FileNotFoundException e) {
      e.printStackTrace();
    } catch (IOException e) {
      e.printStackTrace();
    }

  }

  public static void main(String[] args) throws IOException, ClassNotFoundException,
InterruptedException {
       try {
                    System.out.println("Server  initialized\n");
                    DatagramSocket serverSocket = new DatagramSocket(6666);  //Scoket
Creation
      byte[] incomingData = new byte[1024 * 1000 * 50];
      while (true) {
        DatagramPacket incomingPacket = new DatagramPacket(incomingData,  //Receiegn
pAKET
incomingData.length);
                          long start = System.currentTimeMillis();
                          serverSocket.receive(incomingPacket);
                          long now = System.currentTimeMillis();
                          long ttime = now - start;
                byte[] data = incomingPacket.getData();
        ByteArrayInputStream in = new ByteArrayInputStream(data);
        ObjectInputStream is = new ObjectInputStream(in);
         Files = (Files) is.readObject();
        if (Files.getStatus().equalsIgnoreCase("Error")) {
          System.out.println("No Input");
          System.exit(0);
        }
        createAndWriteFile();   // writing the file to hard disk
                              System.out.println("Recieved file in : "+ttime+" nanosec");
                              InetAddress IPAddress = incomingPacket.getAddress();
        int port = incomingPacket.getPort();

        String reply = "File has been recieved";
        byte[] replyBytea = reply.getBytes();
        DatagramPacket replyPacket =
            new DatagramPacket(replyBytea, replyBytea.length, IPAddress, port);//Creating
Data PACKET a and Loacation Client
        serverSocket.send(replyPacket);  //Sending Packet to client
        Thread.sleep(3000);

      }

    } catch (SocketException e) {
      e.printStackTrace();
    }
  }
}
```