CS550 Advanced Operating Systems
Programming Assignment 3
Source Code

submitted by:
Jayanth Vangari
A20337867

a. PA3.java

```
package PA3;
import java.io.*;
import java.util.*;
import PA3.Location;
import PA3.Lookup;
import PA3.ClientHandler;
//  PA3.java displays the client operations of the peer and allows to select register,search and download operations..
public class PA3
{
	static String rep;
	static int args_len;
	static int serverid;
	public static void main(String[] args) throws Exception
	{
		boolean final_result=false	;
		String val;
		String file;
		args_len=args.length;
		//String rep=null;
		Peer pn=new Peer();
		Lookup lp=new Lookup();
		Location l=new Location();
		serverid=Integer.parseInt(args[0])-1;
		if(args_len>1)
		{
			rep=args[1];
		}
		pn.createpeer(serverid);
		while(true)
		{
			System.out.println("\nFile Operations");
			System.out.println("1. register");
			System.out.println("2. search");
			System.out.println("3. obtain");
			System.out.println("0. exit");
			System.out.println("\nSelect any operation");
			Scanner s=new Scanner(System.in);
			int in=s.nextInt();
			Scanner k=new Scanner(System.in);

			switch(in)
```

```
                {
                case 0:         System.exit(0);
                case 1:         final_result=pn.register();
                                System.out.println(final_result);
                                break;

                case 2:         //System.out.println(final_result);
                                if(final_result==true)
                                {
                                        String search;
                System.out.println("Please provide the file name you wish to lookup");
                                        Scanner s1=new Scanner(System.in);
                                        search=s1.nextLine();
                                        lp=pn.retrieve(search);

                                l=pn.selectpeer(lp.getlookfupfile(),lp.getpeerlist());
                                }
                                else{ System.out.println("register files first");
                                }
                                break;
                case 3:         file=pn.obtain(l);
                                System.out.println(file+"is received");
                                break;
                default:
                                System.out.println("Enter 1,2 or 3");
                }
        }
    }
```

b. Peer.java

```
package PA3;
import java.io.*;
import java.net.*;
import java.util.*;

import org.apache.commons.lang3.RandomStringUtils;
import PA3.Location;
import PA3.Lookup;
import PA3.ClientHandler;
```

```
	//class Peer contains various functionalities of peer
public class Peer implements Runnable
{
	public ServerSocket peerserver;
	public Socket client;
	public int clientid;
	public  static int servport;
	public  static String peerdirpath;
	public String peer;
	static boolean registered=false;
	public static ArrayList<String> name=new ArrayList<String>();
	public String peerdir;
	static int count;
	public  static int[] array_ports;
	static String[][] peerinfo;
	public static Socket csocket[];
	public static File f1;
	//creates a peer and reads config file and connects to all other peers
	void createpeer(int id) throws IOException {
		String k;
		BufferedReader br,br1;
		String currentDir=System.getProperty("user.dir");

		File f=new File(currentDir+"/config");
		try{
			br1=new BufferedReader(new FileReader(f));
			br=new BufferedReader(new FileReader(f));

		int j=0;
		clientid=id;
		while((k=br1.readLine())!=null)
		{
			count++;
		}
		System.out.println("Total Servers : "+count);
		peerinfo=new String[count][4];
		csocket=new Socket[count];
		array_ports=new int[count];
		while((k=br.readLine())!=null)
		{
			StringTokenizer st=new StringTokenizer(k," ");
			int i=0;
			while(st.hasMoreTokens())
```

```
		{
			peerinfo[j][i]=st.nextToken().toString();
			i++;
		}
		j++;
}
servport=Integer.parseInt(peerinfo[id][1]);

}catch(Exception e)
{
		e.printStackTrace();
}
new Thread(new Peer()).start(); //server thread for peer is started
System.out.println("start all the peers and press any key");
Scanner input=new Scanner(System.in);
String read=input.nextLine();
//connections to other peers is made
for(int i=0;i<count;i++)
{
		try {
		csocket[i]=new Socket(peerinfo[i][0],Integer.parseInt(Peer.peerinfo[i][1]));
			array_ports[i]=Integer.parseInt(Peer.peerinfo[i][1]);
		} catch (Exception e) {
			e.printStackTrace();
		}
}
String currentDir1 = System.getProperty("user.dir");
Scanner a=new Scanner(System.in);
peerdir="peer"+(id+1);
peerdirpath=currentDir+"/"+ peerdir;
f1=new File(peerdirpath);
if(f1.exists())
{
		System.out.println("exists");
		String k1=f1.getAbsolutePath();
		System.out.println("directory exists at:"+k1);

}
else{
			if(f1.mkdirs());
			String k1=f1.getAbsolutePath();
			System.out.println("Directory created at:" +k1);
		}
```

```
}
//register functionality of the peer
public boolean register()
{
ObjectOutputStream ops;
ObjectInputStream ips;
File f=null;
int node;
int i=0;
int replica_node=0;
boolean result=false;
String message,send;
String[] parts;
f= new File(peerdirpath);
File[] filename =f.listFiles();
        for(File file: filename)
{
        if(file.isFile())
        {
                name.add(i,file.getName());
                i++;
        }
}

for(i=0;i<name.size();i++)
{

        //hashes the file name to register it at one of the other servers

        node=(int)hash(name.get(i))%count;
        if(node<0)
                node=-(node);
        try{

                ops=new ObjectOutputStream(csocket[node].getOutputStream());
        send=clientid+" "+1+" "+name.get(i)+" "+servport+";"+peerdirpath+";";
                ops.writeObject(send);
                ips=new ObjectInputStream(csocket[node].getInputStream());
                message=(String) ips.readObject();
                if(message.contains("success"))
                {
                        parts=message.split(" ");
                        replica_node=Integer.parseInt(parts[1]);
```

```
                                result=true;
                        }
                        if(result==true && PA3.args_len>1 && PA3.rep.contentEquals("r"))
                        {
                                //replicates files at replica_node if called.
                                replicate(replica_node,i);

                        }

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

        }
        return result;
}

public void replicate(int id,int i)
{
        File f;
        int len=0;
        int n;
        String message;
        boolean result=false;
        byte[] recvfile=new byte[64*1024];
        BufferedOutputStream bos=null;
        byte[] filetosend=new byte[64*1024];
        ObjectOutputStream ops=null;
        ObjectInputStream ips=null;
        DataInputStream dis=null;
        int l=0;

        try{

                String file=peerdirpath+"/"+name.get(i);
                f=new File(file);
                message="r"+" "+name.get(i)+" "+f.length();
                ops=new ObjectOutputStream(csocket[id].getOutputStream());
                bos=new BufferedOutputStream(csocket[id].getOutputStream());
                ops.writeObject(message);
```

```
                                FileInputStream infile=new FileInputStream(f);
                                //transfer file to replica_node
                                while(infile.available()>0)
                                {
                                len=infile.read(filetosend);
                                bos.write(filetosend,0,len);
                                }

                                bos.flush();
                                infile.close();
                                ips=new ObjectInputStream(csocket[id].getInputStream());

                                String reply=(String)ips.readObject();

                        }
                catch(Exception e)
                {}

        }

        String tempinfo;
//allows search operation for a file at other servers
public Lookup retrieve(String key) {
                Lookup l=null;
                int node;
                String message,send;
                node=(int)hash(key)%count;
                if(node<0)
                node=-(node);
                send="c "+2+" "+key;
                try
                {
                ObjectOutputStream ops=new
ObjectOutputStream(csocket[node].getOutputStream());
                ops.writeObject(send);
                ObjectInputStream ips=new ObjectInputStream(csocket[node].getInputStream());
                tempinfo=(String) ips.readObject();
                }

                catch(Exception e)
                {

                }
```

```
            if(tempinfo.equals("FNF"))
            { // file is not registered or present at the indexing server message “FNF” returned
            System.out.println("File not Found");
            System.out.println("Please provide the file name you wish to lookup");
            Scanner be=new Scanner(System.in);
            key=be.nextLine();
            retrieve(key);
            }
            l=new Lookup(key,tempinfo);
            return l;

}
//selects one of the peers if multiple peer list is returned
public Location selectpeer(String file,String list )
{
            String[] parts;
            String path,ipaddress;
            int serverport;
            int in;
            ArrayList<String> substr=new ArrayList<String>();
            int j=0;
                        int k=0;

                        list=list.replaceAll("[^a-zA-Z\\d.\\,\\/\\;\\-\\_]","");


                        for(int i1=0;i1<list.length();i1++)
                        {
                                    if(list.charAt(i1)==',')
                                    {
                                                substr.add(j,list.substring(k,i1));
                                                j++;
                                                k=i1+1;

                                    }
                                    if(i1==(list.length()-1))
                                      {  substr.add(j,list.substring(k,i1+1));
                                      }

                                    }
```

```
                Evaluation e=new Evaluation();
                if(e.len==0)
                {
                        System.out.println("\n PEERS LIST");
                        for(int i=0;i<substr.size();i++)
                        {
                                System.out.println(i+":"+substr.get(i));
                        }
                        System.out.println("Select one of the peers");
                        Scanner select=new Scanner(System.in);
                        in=select.nextInt();

                }
                else
                {       in=0;

                }
                if(in<0 && in>=(substr.size()-1))
                {
                        selectpeer(file,list);
                }

                peer=substr.get(in);
                parts=peer.split(";");
                ipaddress=parts[3];
                serverport=Integer.parseInt(parts[1]);
                path=parts[2];
                Location loc=new Location(file,ipaddress,serverport,path);
                return loc;
        }


        //Hash Function  that hashes the file name and returns a int value representing one of the servers.
private long hash(String key) {

 long hash=65599;
        for (int j=0;j<key.length();j++)
        {
        hash=(key.charAt(j)+hash)*33+j;
        while(hash>Long.MAX_VALUE)
        {
```

```
                hash=hash/10;
        }
        }
        return hash;
        }
        //downloads file at client
public String obtain(Location locate)
        {
                Lookup ll=new Lookup();
                Location loc=new Location();
                String[] parts;
                String send,length=null;
                int id=0,c1=0;
                long len=0;
                String saddress=locate.getsaddress();
                int sport=locate.getportadd();
                String filesearched=locate.getfname();
                String path=locate.getpath();
                String name,ext;
                File file=null;
                FileOutputStream fops=null;;
                ObjectOutputStream reqfile=null;
                BufferedInputStream bis=null;
                ObjectInputStream ois=null;
                OutputStream os=null;

                send="c "+3+" "+filesearched+" "+path;

                for(int i=0;i<count;i++)
                        {
                                if(sport==array_ports[i])
                                {
                                        id=i;
                                }
                        }

                try
                {
                        reqfile= new ObjectOutputStream(csocket[id].getOutputStream());
                        reqfile.writeObject(send);
                        ois=new ObjectInputStream(csocket[id].getInputStream());
                        length=(String)ois.readObject();
                }
```

```
catch(Exception ee)
{       //if no connection , try connect and select the replica node
        while(!(csocket[id].isConnected()) && count<=3)
        {
                try
                {
csocket[id]=new Socket(peerinfo[id][0],Integer.parseInt(Peer.peerinfo[id][1]));
                count=count+1;
                }
                catch(Exception ie)
                {}

        }
                System.out.println("select the neighour node in the peer list");
                ll=retrieve(filesearched);

                loc=selectpeer(ll.getlookfupfile(),ll.getpeerlist());
                obtain(loc);
                System.out.println(filesearched+" received");
                System.exit(0);


}
byte[] recvfile=new byte[64*1024];
//file name given to the downloaded file
if(filesearched.contains("."))
{
        parts=filesearched.split("\\.");
        name=parts[0];
        ext=parts[1];
        if(name.contains("_cpy"))
        {
                name=name.concat("y");
                filesearched=(name+"."+ext).toString();
                file=new File(peerdirpath+"/"+filesearched);
        }
        else
        {
                name=name+"_cpy";
                filesearched=(name+"."+ext).toString();
                file=new File(peerdirpath+"/"+filesearched);
        }
```

```
}
else
{
        if(filesearched.contains("_cpy"))
        {
        filesearched=filesearched.concat("y");
        file=new File(peerdirpath+"/"+filesearched);
        }
        else
        {
                filesearched=filesearched.concat("_cpy");
                file=new File(peerdirpath+"/"+filesearched);
        }
}
try
{
        //download the file
        os=new FileOutputStream(file);
        InputStream instream=csocket[id].getInputStream();
        bis=new BufferedInputStream(instream);
        while((bis.available())>=0)
        {
                c1=bis.read(recvfile);
                len=len+c1;
                os.write(recvfile,0,c1);
                if(Long.parseLong(length)==len)
                {
                        os.flush();
                        break;
                }
        }
}
catch(Exception e)
{}
finally
{
        try
        {
                os.flush();
                fops.close();

        }
        catch(Exception e){}
```

```
        }
        return filesearched;
        }

        public void run()
        {
                Peer pe=new Peer();
                try
                {       //create server socket
                        String k;
                        peerserver=new ServerSocket(servport);
                        System.out.println("server started at port:"+servport);
                        while(true)
                        {       //accept connections from other clients
                                Socket peerclient1=peerserver.accept();
                                if(peerclient1!=null)
                                {
                                        Peer s= new Peer();
                                        s.client=peerclient1;
                                }
                                ClientHandler ch=new ClientHandler();
                                ch.find(peerclient1);
                        }
                }
                catch(Exception ie)
                {
                        ie.printStackTrace();
                }
        }
}
//server functionality of the peer
```

c. ClientHandler.java

```
package PA3;
import java.io.*;
import java.net.InetAddress;
import java.net.Socket;
import java.util.*;
class ClientHandler implements Runnable
 {
         File f;
        byte[] filetosend=new byte[64*1024];
        int len=0;
```

```
        //Hashtable with keys as filenames and peers information as values with initially capacity is created
        public static Hashtable<String,ArrayList<String>> ht=new
        Hashtable<String,ArrayList<String>>(4000003);
        ArrayList<String> ll;
        Peer pe=new Peer();
        String file,reqfile,path,name;
         private Socket client;
         public ClientHandler()
         { }
         public ClientHandler(Socket client)
         {
                this.client=client;
         }
        public void find(Socket client)
        {
                 new Thread(new ClientHandler(client)).start();
        }

        @Override
         public void run()
         {
                byte[] bytearray=new byte[64*1024];
                ObjectInputStream ois;
                ObjectOutputStream ops;
                BufferedOutputStream bos;
                FileOutputStream fops=null;;
                OutputStream os=null;
                BufferedInputStream bis=null;
                DataOutputStream dos=null;
                Peer pe=new Peer();
                String file;
                String length=null;
                int c1=0,len;

                InetAddress addr=client.getInetAddress();
                String ipaddress=addr.getHostAddress();
                while(true)
                {
                        try
                        {
                                ois = new ObjectInputStream(client.getInputStream());
                                ops=new ObjectOutputStream(client.getOutputStream());
```

```
String message=(String)ois.readObject();
byte[] recvfile=new byte[64*1024];
long l=0;

String[] parts=message.split(" ");
//receives the file to replicate from other client and stores it in its directory
if(parts[0].contentEquals("r"))
{
	f=new File(pe.peerdirpath+"/"+parts[1]);


	try{
		os=new FileOutputStream(f);
		InputStream instream=client.getInputStream();
		bis=new BufferedInputStream(instream);
	//downloads the file
		while((bis.available())>=0)
		{	c1=bis.read(recvfile);
			l=l+c1;
			os.write(recvfile,0,c1);
			if(Long.parseLong(parts[2])==l)
			{
				os.flush();
				break;
			}

		}
		ops.writeObject(""+l);

	}


	catch(Exception e)
	{

	}
}
//does the PUT opertion on hashtable  for registering files
if( Integer.parseInt(parts[1])==1)
{
	int replica=Integer.parseInt(pe.peerinfo[pe.clientid][2]);
	if((Integer.parseInt(parts[0]))==replica)
	{
```

```
                        if((Integer.parseInt(parts[0]))==0)
                                replica=(Integer.parseInt(parts[0]))+1;
                        else
                                replica=replica-1;
                }
                parts[3]=replica+";"+parts[3].concat(ipaddress);

                if(ht.containsKey(parts[2]))
                {
                        if(!ht.get(parts[2]).contains(parts[3]))

                                ht.get(parts[2]).add(parts[3]);
                }
                else
                {
                        ll=new ArrayList<String>();
                        ll.add(parts[3]);
                        ht.put(parts[2],ll);
                }
                ops.writeObject("success"+" "+replica);
        }
//retrieve operation on hash table when file is searched

        else if(Integer.parseInt(parts[1])==2)
        {
                String value;
                if(ht.containsKey(parts[2]))
                {
                        value=(ht.get(parts[2])).toString();
                        ops.writeObject(value);
                }
                else
                {
                        value="FNF";
                        ops.writeObject(value);
                }
        }
//sends file when download request is received
        else if(Integer.parseInt(parts[1])==3)
        {
                name=parts[2];
                path=parts[3];
                file= path+"/"+name;
```

```
                                        f=new File(file);
                                        ops.writeObject(""+f.length());
                                        bos=new BufferedOutputStream(client.getOutputStream());
                                        FileInputStream infile=new FileInputStream(f);
                                        while(infile.available()>0)
                                        {       len=infile.read(filetosend);
                                                bos.write(filetosend,0,len);
                                        }
                                        bos.flush();


                                        infile.close();

                                }
                        }
                        catch(EOFException eof)
                        {}
                        catch (Exception e) {}
                }

        }
}
```

d. Lookup.java

```
//class Lookup contains a constructor with lookup filename and peer list of the file
// it is the return type of method lookup()
package PA1;
class Lookup{
                  String lookupfile;
                  String peerlist;
                public Lookup(String lookupfile,String peerlist)
                {
                        this.lookupfile=lookupfile;
                        this.peerlist=peerlist;
                }
                public Lookup() {  }
                public String getlookfupfile() //returns lookup file
                {
                        return this.lookupfile;
                }
                public String getpeerlist()
                {
                        return this.peerlist; //returns peerlist
```

```
        }
    }
```

e. Location.java

```
package PA1;
import java.io.*;
// Location contains filename , peer address and port of that peer
//it is the return type of method selectpeer()
    public class Location
    {
        String file;
        String saddress;
        int serverport;
        String path;



        public Location()
        {

        }
//Location class constructor
        public Location(String file,String saddress,int serverport)
        {
            this.file=file;
            this.saddress=saddress;
            this.serverport=serverport;
            this.path=path;
        }

        public String getfname()
        {
            return this.file; //returns file name
        }
        public String getsaddress()
        {
            return this.saddress; // returns peer address

        }
        public int getportadd()
        {
            return this.serverport; //returns peer port
        }
```

```
        String getpath()
        {
                return this.path; //returns directory path of the file
        }

}
```

f. Evaluation.java

```
package PA3;
import java.io.*;
import java.net.UnknownHostException;
import java.util.ArrayList;
import java.util.Scanner;

import org.apache.commons.lang3.RandomStringUtils;

import PA3.Lookup;

 //Evaluation class runs evaluation on the peers by doing varying size of operations of various file sizes as specified by user by creating n files and does ‘n’ register operation and doing  ‘n’ search and ‘n’ download operations and measuring respoonse time

class Evaluation {
        public long starttime,endtime ;
       // static long fortime;
        ArrayList<String> fnames=new ArrayList<String>();
        ArrayList<Lookup> peerslist=new ArrayList<Lookup>();
       ArrayList<Location> loca=new ArrayList<Location>();
        String dirpath;
        static Peer peers=new Peer();
        boolean register=false;
        Location location=new Location();
        Lookup ll=new Lookup();
        static String peernum;
       static int len=0;
       public static void main(String[] args) throws Exception
       {       len=args.length;
               Evaluation e=new Evaluation();
               peernum=args[0];      //enter through command line the id for the peer
```

```
                peers.createpeer(Integer.parseInt(peernum)-1);
                        e.eval(args[1],args[2]); // arg[1] - filesize , arg[2] - number of files
                        System.exit(0);

        }
        //evaluates register, search and download operations for different file sizes and
varying number of files
        private void eval(String filesize,String filenum) throws Exception
        {
                Peer a=new Peer();
                Evaluation ee=new Evaluation();
                int exival=2;
                // creates random files by executing script ./testfiles.sh
                String[] env = {"PATH=/bin:/usr/bin/"};
                String cmd ="./testfiles.sh"+" "+filenum+" "+filesize+" "+peernum;
                ///test$nodeid_filesize_$filenum
                Process process =Runtime.getRuntime().exec(cmd, env);
                try {
                        exival=process.waitFor();
                } catch (Exception e) {
                        e.printStackTrace();
                }
                if(exival==0)
                {
                while(true)
                {       //allows user to choose select '1' in the start and other operations in
chronologically
                        System.out.println("\nChoose options in sequence:");

                        System.out.println("1. get register time");
                        System.out.println("2. get search time");
                        System.out.println("3. get download time");
                        System.out.println("0. exit");
                        Scanner in=new Scanner(System.in);
                        int i=in.nextInt();
                        switch(i)
                        {
                        // the time taken for register
                        case 1: ee.getregtime(filesize,filenum);
                                break;
                        // the time taken for search
                        case 2: ee.getsearchtime(filesize,filenum);
```

```
                    break;
                // the time taken for downloading files
                case 3: ee.getdownloadtime(filesize);
                    //delete files in the peer directory after evaluation
                    System.out.println("delete files? Press any key..");
                    Scanner input=new Scanner(System.in);
                    String read=input.nextLine();
                    String currentDir=System.getProperty("user.dir");
                    String dirpath=a.peerdirpath;
                    File file=new File(dirpath);
                    File[] files = file.listFiles();
                    for (File f: files)
                    {
                        f.delete();
                    }
                    System.out.println("deleted.");

                    break;
                    //exit
                case 0: System.exit(0);
                    break;
                default: System.out.println("Enter 1,2 or 3");

                }
            }
            }
    }


private void getregtime(String filesize,String filenum) throws UnknownHostException,
IOException, ClassNotFoundException
    {
        starttime=System.currentTimeMillis();
        register=peers.register(); //call for register method to register files
        endtime=System.currentTimeMillis();
        System.out.println("\nTime elapsed to register"+" "+filesize+" files:"+
(endtime-starttime) + " millisecs");


    }
```

```
	private void getsearchtime(String filesize,String filenum) throws Exception {
		Peer pe1=new Peer();
		String name;
		int peerid=Integer.parseInt(peernum);
		if(pe1.count!=1)
			peerid=Integer.parseInt(peernum)+1;
		if(register==true)
		{		starttime=System.currentTimeMillis();
			for (int i=1;i<=Integer.parseInt(filenum);i++)
			{
				if((peerid==Integer.parseInt(peernum)) && (pe1.count!=1))
				{
					peerid++;
				}

				if(peerid>pe1.count)
				{
					peerid=1;
					if(Integer.parseInt(peernum)==1 && pe1.count!=1)
					{ peerid++;
					}
				}
					name="test"+peerid+"_"+filesize+"_"+i+".BIN";
					//System.out.println(name);
				peerid++;
				//callretrieve method to search files at servers
				peerslist.add((peers.retrieve(name)));

			}
			endtime=System.currentTimeMillis();
			System.out.println("Time elapsed to search"+" "+filesize+" files:"+
(endtime-starttime)+" millisecs");

		}

	}
private void getdownloadtime(String filesize) throws IOException {
	String fname;
	String peerlist;

	for(int i=0;i<peerslist.size();i++)
	{		fname=(String)peerslist.get(i).getlookfupfile();
```

```
                peerlist=(String)peerslist.get(i).getpeerlist(); //peerlist with peer info for
searched files
                loca.add(i,peers.selectpeer(fname,peerlist));
        }
        starttime=System.currentTimeMillis();
        for(int i=0;i<loca.size();i++)
        {
                peers.obtain(loca.get(i)); //call obtain() method to download files

        }
        endtime=System.currentTimeMillis();
        System.out.println("Time elapsed to download"+" "+filesize+"  files:"+
(endtime-starttime)+" millisecs");


}
}
```